Tampereen yliopisto

Jorma Syrjä

TÖRMÄYSTEN ENNUSTAMINEN OBJEKTIEN SEURANNALLA

Tieto- ja sähkötekniikan tiedekunta
Diplomityö
Toukokuu 2019

TIIVISTELMÄ

Jorma Syrjä: Törmäysten ennustaminen objektien seurannalla
Diplomityö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2019

Tässä työssä toteutettiin sovellus, joka pyrkii tunnistamaan ja seuraamaan videosta ihmisiä ja heidän liikkeitään. Sovellus ennustaa seurattujen ihmisten liikeratoja ja yrittää havaita tilanteet, joissa kaksi tai useampi ihmistä on tormäämässä.

Sovelluksen toteuttamiseen käytettiin Python-ohjelmointikieltä sekä OpenCV-kuvankäsittelykirjastoa. Sovelluksen toiminta jaettiin kolmeen osaan, kohteiden tunnistus, seuranta ja törmäysten ennustus.

Kohteiden tunnistukseen käytettiin siihen soveltuvaa, erittäin hyväksi todettua Yolo-neuroverkkoa. Seurannassa hyödynnettiin OpenCV:n tarjoamia toteutuksia seuraavista seuranta-algoritmeista: kernelisöity korrelaatiosuodatin, diskriminatiivinen korrelaatiosuodatin, MOSSE, MedianFlow. Seuranta-algoritmeja vertailtiin testivideosekvenssillä, jossa algoritmin suorituskykyä mitattiin yhden seurattavan kohteen avulla laskemalla IoU, Precision, Recall sekä päivitysnopeus.

Algoritmien vertailutuloksista tehtiin johtopäätös, että MOSSE-algoritmi on erittäin hyvä valinta nopeuden ja tarkkuuden ansiosta. Sitä tarkempi on ainoastaan diskriminatiivinen korrelaatiosuodatin, joka on seuranta-algoritmeista hitain. Valintapäätös tulee tehdä käyttötapauksen mukaan, jos kyseessä on turvallisuuteen liittyvä sovellus, niin tarkkuus on tärkeämpää kuin nopeus.

ABSTRACT

Jorma Syrjä: Collision Prediction using Object Tracking
Master's Thesis
Tampere University
Master's Degree Programme in Information Technology
May 2019

This master's thesis describes an implementation of a simple application whose goal is to detect and track people from an input video and also predict their trajectories and possible collisions using tracking information.

The application was implemented in Python programming language and OpenCV graphical processing library. The implementation was divided into three parts: object detection, tracking and collision prediction.

Object detection was done using state-of-the-art neural network Yolo. Tracking was done using OpenCV's implementations for tracking algorithms kernelized correlation filter, discriminative correlation filter, MOSSE and MedianFlow. Each of these algorithms were evaluated using a test video sequence with one tracked object, from which IoU, Precision, Recall and update speed were calculated.

From the evaluation results it was concluded that the MOSSE algorithm is an excellent choice because of its speed and accuracy. It is only surpassed in accuracy by the discriminative correlation filter algorithm, which is the slowest of the four. Therefore, the decision what algorithm to use for tracking should be decided depending on the use case. In cases where security is involved, accuracy is more important than speed.

ALKUSANAT

Tämä diplomityö toteutettiin Wapice Oy:lle.

Kiitän Ilari Kampmania diplomityön aiheesta ja professori Heikki Huttusta työn ohjaamisesta.

Tampereella 9.5.2019

Jorma Syrjä

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

CPU	Central Processing Unit eli prosessori
GPU	Graphical Processing Unit el näytönohjain
IoU	Intersection over Union
Precision	Tarkkuus
Recall	Herkkyys
Python	Ohjelmointikieli
TP	True Positive eli oikea positiivinen näyte
TN	True Negative eli oikea negatiivinen näyte
FP	False Positive eli väärä positiivinen näyte
FN	False Negative eli väärä negatiivinen näyte
FPS	Frames Per Second eli ruudunpäivitysnopeus
Ground truth	Suora havainto
NMS	Non-maximum suppression eli rajoitus

1. JOHDANTO

Koneoppinen, konenäkö ja tekoäly ovat käsitteitä, jotka ovat nousseet viimeaikoina esiin yhä useammassa kontekstissa. Etenkin tekoälyn merkitys ja hyödyntäminen puhuttavat paljon muidenkin kuin tietotekniikan alojen asiantuntijoita. Termiä tekoälyä käytetään usein sanan muodikkuuden takia, vaikka oikeasti kyseessä on jokin koneoppimiseen liittyvä asia. Koneoppimseen liittyvä tekoäly myös usein virheellisesti rinnastetaan ihmisiltä näyttäviin robotteihin, kuten suosituissa elokuvissa.

Koneoppimista on tutkittu jo noin 70-luvulta asti. Tietokoneiden ja laskentatehon kehittymisen ansiosta on kehitetty uusia algoritmeja ja tekniikoita, kuten neuroverkot, jotka ovat erityisesti kuumia puheenaiheita. Kehityksen ja käyttöönoton helpottumisen myötä yritetään jatkuvasti keksiä uusia käyttötapauksia, jotta voidaan sanoa, että jokin järjestelmä toimii tekoälyä hyödyntäen. Hyvin usein tällaisissa tapauksissa koneoppimista tai tekoälyä ei tarvita ja oikeastaan sen käyttäminen vain lisää hämmennystä.

Mikä on siis koneoppimisen ja tekoälyn ero? Ensinnäkin, koneoppiminen tutkimusalana on yksi osa tekoälyn tutkimusalaa. Toiseksi, koneoppimisen soveltamisessa on usein kyse sellasesta järjestelmästä, joka syötteen perusteella yrittää tuottaa ulostulon. Tämä poikkeaa hieman tekoälyn soveltamisesta, missä on usein kyse järjestelmästä, joka tuottaa päätöksiä, esimerkiksi toiminnon, joka pysäyttää robotin. Kolmantena erona voisi mainita tavoitteet, jotka ovat hyvin erilaisia. Koneoppimisessa tavoitteena on oppia asioita datasta, esimerkiksi toistuvia kaavoja. Tekoälyssä tavoitteena on järjestelmä, joka toimii itsenäisesti ja usein imitoiden ihmisen älyä ja päätöksentekokykyjä.

Robottien tekoäly ja itsenäisyys niiden käyttötarkoitusten ulkopuolella lienee kaukana tulevaisuudessa. Niiden merkittävin heikkous on aistien puute, kuten näkö, kosketus ja kuulo. Niitä korvaamaan luodut sensorit aiheuttavat riippuvuuksia, jotka ovat myös heikkouksia. Nämä heikkoudet korvaa kuitenkin prosessori, joka pystyy laskemaan ja tekemään päätöksiä mikrosekunneissa eli ajassa, johon yksikään ihminen ei kykene. Olisi siis hyvin kannattavaa korvata ihminen koneella aina, kun se on mahdollista. Mutta miten kone voi pärjätä ihmisen aisteille?

Konenäkö on kyseessä silloin, kun järjestelmään viedään kuvia tai videoita, joissa sijaitsevaa tietoa käytetään hyväksi jollain tavalla. Esimerkiksi virvoitusjuomatehtaan

tuotantolinjalla voitaisiin kuvata pulloissa nesteen pinnan korkeutta ja suorittaa sitä kautta laadunvalvontaa poistamalla ne pullot, joissa nestettä on liian vähän.

Konenäössä voidaan hyödyntää koneoppimisen kouluttamista. Kasvojen havaitsija koulutetaan näyttämällä sille kuvia kasvoista, joiden avulla se oppii kasvojen peruspiirteet, esimerkiksi silmät, nenän ja suun. Koulutettua havaitsijaa voidaan käyttää esimerkiksi kamerasovelluksessa, joka automaattisesti korostaa havaittuja kasvoja paremman lopputuloksen saavuttamiseksi. Tässä on tärkeää huomata ero kasvojen**tunnistukseen,** jossa on usein tavoitteena selvittää myös se tieto, kenelle kasvot kuuluvat.

Konenäköä on käytetty useissa muodoissa esimerkiksi teollisuudessa jo 90-luvulta asti. Modernin tekoälybuumin ansiosta on alettu miettimään, miten koneoppimista, konenäköä ja tekoälyä voitaisiin hyödyntää esimerkiksi korvaamalla jokin yksitoikkoinen ihmiselle tarkoitettu työtehtävä robotilla. Esimerkki yksitoikkoisesta työstä voisi olla tarkistus, jossa varmistetaan, että osat ovat paikoillaan tietyssä vaiheessa kokoonpanoa.

Aika usein kuitenkin todetaan, että nykyiset ratkaisut ovat liian epäluotettavia ja että ihmisen pitää olla ainakin valvomassa automaattista tekoälyyn perustuvaa ratkaisua. On siis vielä liian aikaista pelätä, että tekoäly korvaa ihmisten työt muissa kuin yksinkertaisissa kokoonpanolinjatöissä.

Joskus konenäköä verrataan virheellisesti ihmisen näköön. Vaikka molemmissa tapauksissa kyse on visuaalisesta datasta, jonka perusteella tehdään päätöksiä, on konenäön käyttötapaukset usein hyvin erilaisia. Tämä johtaa siihen, että konenäköä hyödyntävään järjestelmään ei aina ole hyödyllistä syöttää samanlaista dataa kuin ihmiselle. Konenäkösovelluksesta riippuen kuvasta voidaan esimerkiksi poistaa värit tai muita piirteitä, tai kuva voidaan ottaa ultraviolettikameralla. Toisin sanoen konenäköä hyödyntävään järjestelmään syötettävän datan ei välttämättä tarvitse olla ihmisen ymmärrettävissä.

Myös turvallisuusalalla on mietitty mahdollisia käyttötapauksia tekoälylle. Nykyisillä ratkaisuilla on jo mahdollista luoda kasvoja tunnistava neuroverkko, jota voidaan käyttää esimerkiksi varmistamaan, että tehdasalueelle ei pääse ulkopuolisia. Toinen esimerkki on Kiina, jossa viranomaiset hyödyntävät kasvojentunnistusta laajasti ihmisten seurannassa [1]. Lienee realistista sanoa, että yhä useampi ihmisen suorittama valvontatyö tullaan automatisoimaan ainakin osittain tulevaisuudessa.

Tässä diplomityössä tarkastellaan tapausta, jossa valvontakamerasta saadusta videokuvasta yritetään havaita ja ennustaa mahdolliset törmäystilanteet. Idea on peräisin tehtaan varastotilasta, jossa kulkee useita työntekijöitä ja mahdollisesti myös työkoneita

ja tavaraa kuljettavia trukkeja. Tämä työ on rajattu ainoastaan ihmisten törmäysten ennakoimiseen. Lisäksi reaaliaikaisen valvontakamerakuvan saatavuuden vaikeuden takia työssä käytetään videokuvaa.

Tavoitteena on kehittää ohjelma, jolle annetaan parametriksi kovalevyllä sijaitsevan videon sijainti, videoon kuuluva kohdekuva (esimerkiksi huoneen pohjasuunnitelma) ja videon ensimmäisen kehyksen ja kohdekuvan vastinpisteet sisältävän tiedoston sijainti. Ohjelma lataa tiedostot muistiin, laskee videon ensimmäisen kehyksen ja kohdekuvan välisen homografian ja alkaa käsitellä videota yksi kehys kerrallaan. Käsittelyvaiheessa ohjelma yrittää tunnistaa ihmisiä videokuvasta, seurata heidän liikeratojansa ja ennustaa, onko kaksi tai useampi ihminen törmäämässä toisiinsa. Käsittelyvaihe on raskas, joten ohjelman tulisi vastaanottaa käynnistysparametrina käsittelyvaiheiden toistumistiheys, joka on n kertaa sekunnissa. Ohjelman ajon aikana näytetään kuvaa syötevideosta, johon on merkitty tunnistetut ja seuratut kohteet. Törmäystapahtuman ennustuksen sattuessa ohjelman suoritus pysäytetään ja käyttäjälle näytetään kohdekuva, johon on piirretty ennustetun törmäyksen sijainti.

Työn tuloksena syntyi Python-ohjelma, joka tutkii syötevideota. Ihmiset tunnistetaan ja seurataan ja ne merkitään videokuvaan laatikoiksi. Tietyn ajan välein ohjelma tarkastelee seurattujen ihmisten liikehistoriaa ja yrittää arvata heidän seuraavan sijaintinsa sekunnin kuluttua. Jos kahden tai useamman ihmisen arvioidut sijainnit leikkaavat, ohjelma pysähtyy ja piirtää törmäyskohdan kohdekuvaan.

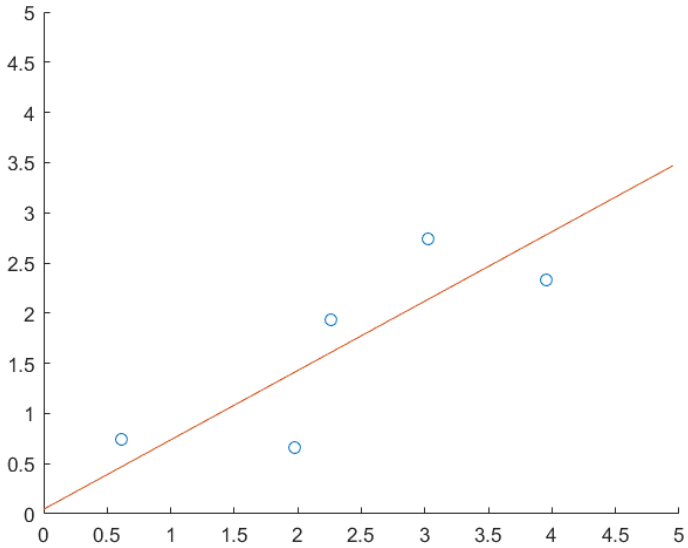
Kappaleessa 2 esitellään lyhyesti työhön liittyvä teoria, muun muassa koneoppiminen, neuroverkot, objektien tunnistus ja kuvan perspektiivimuunnos. Kappaleessa 3 käsitellään objektien seurantaa, mikä on keskeisin asia tässä diplomityössä. Kappaleessa 4 käydään läpi perusteellisesti työn aika toteutettu sovellus, joka seuraa ihmisiä ja yrittää ennustaa mahdolliset törmäystilanteet. Kappaleessa 5 arvioidaan työkaluja ja ratkaisuja, joita toteutetussa sovelluksessa käytettiin. Viimeisenä kappaleessa 6 käydään läpi tulokset ja jatkokehitysideat.

2. TEORIA

1.1 Koneoppiminen

Koneoppiminen on tekoälyn osa-alue, jossa pyritään muodostamaan koulutusdataa kuvaava matemaattinen malli. Muodostuessaan malli oppii piirteitä koulutusdatasta. Mitä paremmin malli yleistää koulutusdatan, sen paremmin se suoriutuu tilanteissa, joissa siihen syötetään koulutusdatasta poikkeavaa dataa.

Yksinkertaisin esimerkki matemaattisen mallin muodostuksesta on suora, joka sovitetaan koordinaatiston pisteisiin (kuva 1). Koulutusdatana toimii siis x,y -pistejoukko ja malli $ax + b$, josta ratkaistaan a ja b.



Kuva 1. (x,y) -pistejoukko, johon on sovitettu suora

Luokittelussa koulutusdataan merkitään luokka, johon kyseinen data kuuluu. Koulutuksen jälkeen luokittajalle voidaan syöttää ei-koulutusdataan kuuluva data, jonka luokittelija arvaa kuuluvan johonkin niistä luokista, jotka koulutusdatassa esitettiin. Arvaukset esitetään todennäköisyytenä [0,1], joista suurin todennäköisyys yleensä valitaan luokittelijan ulostuloksi.

Lineaarisessa regressiossa koulutusdataan x lisätään tieto siitä, minkä arvon y mallin $f(x)$ pitäisi tuottaa. Toisin sanoen yritetään kuvata riippumattoman muuttujan x suhdetta riippuvaan muuttujaan y. Edellämainittu $ax + b$ on esimerkki lineaarisesta regressiosta, jossa etsitään parhaimmat arvot a:lle ja b:lle siten, että keskimääräinen virhe suoran ja pisteiden (x,y) välillä on mahdollisimman pieni. Kyseessä on siis **tappiofunktio**, jota

yritetään minimoida. Eräs käytetty tappiofunktio on keskimääräinen neliövirhe, joka on muotoa:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

missä n on pisteiden lukumäärä, Y_i on alkuperäisen datan y-koordinaatti ja $\hat{y}_i$ on y-koordinaatti sovitetulta suoralta.

Koulutusvaihe voidaan tehdä eri tavoilla. Jos saatavilla oleva data ei sisällä tietoa sen ulostulosta, mallin oppimista kutsutaan **ohjaamattomaksi**. Yleisemmässä tapauksessa ulostulotieto on saatavilla, jolloin kyse on **ohjatusta oppimisesta**. Jos suuressa osassa koulutusdataa ei ole ulostulotietoa, voidaan koulutus suorittaa **osittain ohjatusti.** Yksi tapa suorittaa osittain ohjattu koulutus on tehdä koulutus ensin valvotusti, minkä jälkeen koulutettuun malliin syötetään merkitsemätöntä koulutusdataa. Sen jälkeen voidaan luodaan uutta koulutusdataa niistä ulostuloista, jotka ylittävät tietyn luotettavuusrajan. Uusi koulutusdata lisätään osaksi vanhaa koulutusdataa ja koulutusvaihe suoritetaan uudelleen.

Koulutusvaiheen jälkeen on syytä tarkistaa, onko tuloksena saatu malli hyvä. Yleensä osa saatavilla olevasta koulutusdatasta jätetään sivuun ja käytetään myöhemmin mallin testaamiseen, esimerkiksi 80% datasta käytetään koulutukseen ja loput 20% testaamiseen koulutusvaiheen jälkeen. Jakaminen tulisi tehdä niin, että mahdollisimman paljon vältettäisiin tilannetta, että testidatassa on paljon yhtäläisyyksiä koulutusdatan kanssa.

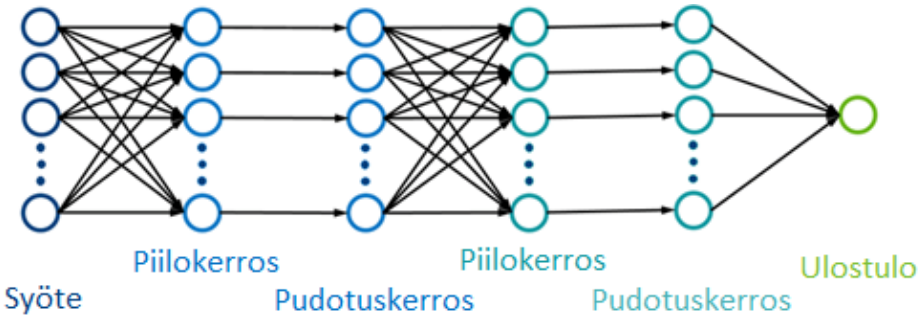
Huonosti tehty koulutusdatan jako tai mallin parametrit voivat johtaa **ylisovittamiseen.** Ylisovittunut malli on saanut liikaa vaikutteita jostain tietystä koulutusdatasta tai datan ominaisuudesta. Tämä tarkoittaa siis sitä, että muodostettu malli kuvaa ainoastaan koulutusdataa sitä yleistämättä. Toisaalta liian tiukat parametrit voivat johtaa **alisovittamiseen,** jolloin malli ei kuvaa koulutusdataa eikä osaa yleistää ongelmaa, jota yritetään ratkaista.

Yksi tapa mallin ja koulutusdatan soveltuvuuden testaamiseen on **ristiinvalidointi,** jossa koulutusdata jaetaan testi- ja koulutusosuuksiin useaan kertaan. Jokaisessa jaossa testi- ja koulutusdata ovat eri osajoukkoja alkuperäisesta koulutusdatasta. Yhden jaon aikana malli koulutetaan ja testataan uudelleen. Testitulokset kirjataan ylös vertailua varten. Testien tarkkuuksien keskimääräinen tarkkuus paljastaa mallin soveltuvuuden datan kanssa. Jos varianssi on suuri, kyseinen malli ei ole hyvä ratkaisemaan ongelmaa. [13]

1.2 Neuroverkot

Neuronien nimitys ja toimintaperiaate perustuvat aivoissa oleviin biologisiin neuroneihin ja niiden muodostamiin neuroverkkoihin. Biologiset neuronit ovat tietojenkäsittelyn näkökulmasta yksiköitä, jotka reagoivat ärsykkeisiin ja sekä vastaanottavat että lähettävät tietoa kemiallisilla ja sähköisillä signaaleilla. Keinotekoiset neuroverkot ovat toiminnaltaan yksinkertaisempia, mutta perustuvat samalla tavalla keinotekoisiin neuroneihin kuin aivot biologisiin neuroneihin.

Neuroverkot muodostuvat kerroksista, joissa on rinnakkaisia neuroneja. Kerrosten välissä on synapseja, jotka yhdistävät neuronit seuraavan kerroksen neuroneihin. Syöte- ja ulostulokerrosten välisiä kerroksia kutsutaan piilokerroksiksi (eng. hidden layer). Ylisovitusta pyritään estämään mm. pudotuskerroksilla (eng. dropout), jotka katkaisevat satunnaisesti neuronien välisiä yhteyksiä.



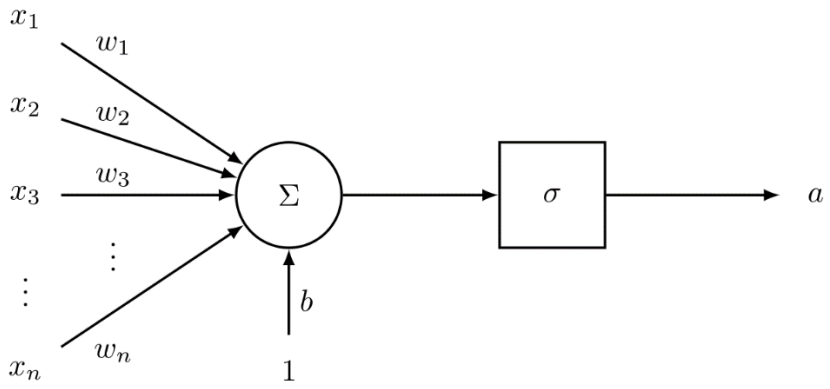
Kuva 2. Esimerkki tiheästä neuroverkon rakenteesta

Tiheässä neuroverkossa jokainen neuroni on yhteydessä seuraavan kerroksen jokaiseen neuroniin (kuva 2). Koulutusvaiheessa yhteyksille määritellään painokertoimet. Painokertoimet w_n ja säätövakio b lasketaan yhteen, ja summa syötetään aktivointifunktioon σ, jonka ulostulo päättää neuronin ulostulon (kuva 3).

Aktivointifunktiona voisi esimerkiksi olla logistinen sigmoid -funktio, joka määritellään seuraavasti:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Neuronin ulostulo saadaan siis laskemalla yhteen syötteet kerrottuna niiden painoilla ja syöttämällä saatu summa aktivointifunktioon.



Kuva 3. Keinotekoisen neuroverkon neuroni [13]

Neuroverkon koulutusvaihe suoritetaan yhdessä tai useammassa aikakaudessa. Yhden aikakauden aikana neuroverkolle näytetään kaikki koulutusdata, usein osajoukkoina. Yhden koulutusdataosajoukon näyttämisen aikana jokaisen neuronin painokertoimet lasketaan uudelleen jollain tietyllä iteratiivisella laskukaavalla.

Koulutuksen alussa neuronien painoille w_n annetaan satunnaiset arvot. Koulutuksessa käytetään niin sanottua vastavirta-algoritmia (eng. back-propagation), joka tarkoittaa sitä, että neuroverkon oikeassa reunassa eli ulostulossa saatu virhe vyörytetään taaksepäin painojen uudelleenlaskemiseksi. Virheellä tarkoitetaan sitä, kuinka paljon verkon ulostulo $\hat{y}$ poikkesi odotetusta ulostulosta y annetulla koulutusdatalla. Virheen suuruutta lasketaan tappiofunktiolla, joka voisi olla esimerkiksi keskimääräinen ulostulojen välinen virhe, kun käydään läpi n kappaletta koulutusdataa:

$$Virhe = \frac{1}{2n}\sum_{n}(\hat{y} - y)^2$$

Tappiofunktion arvo pyritään saamaan mahdollisimman pieneksi säätämällä jokaisen neuronin jokaista painoa. Yksi tapa laskea painolle uusi arvo on gradienttimenetelmä, jossa lasketaan tappiofunktion osittaisderivaatta painon w_{ij} suhteen:

$$\frac{dE}{dw_{ij}}$$

Jos derivaatan arvo on positiivinen, w_{ij} kasvattaminen kasvattaa tappiofunktion arvoa ja jos se on negatiivinen, w_{ij} kasvattaminen pienentää tappiofunktion arvoa. Menetelmää varten tulee määrittää oppimisnopeusparametri $\eta > 0$, joka kerrottaessa derivaatan arvon kanssa varmistaa sen, että w_{ij} muuttuessa tappiofunktion arvo pienenee.

$$w_{ij} = w_{ij} - \eta\frac{dE}{dw_{ij}}$$

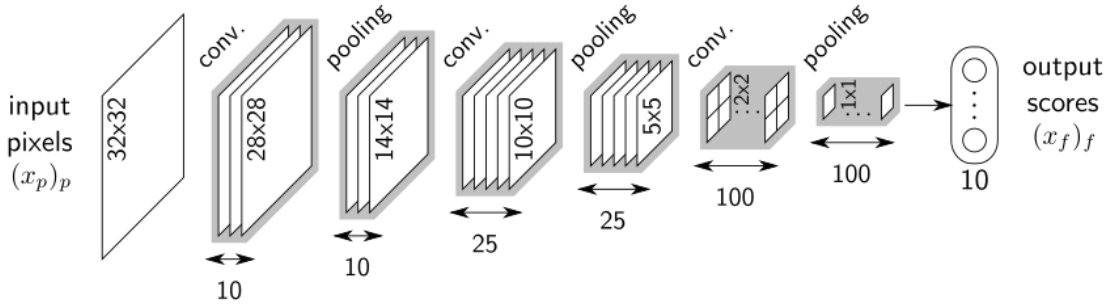
[13]

1.3 Syväoppiminen

Syväoppimisella tarkoitetaan sellaisia neuroverkkoja, joissa on huomattavasti enemmän kerroksia, monimutkaisempia yhteyksiä ja aktivointifunktioita kuin perinteisissä tiheissä neuroverkoissa.

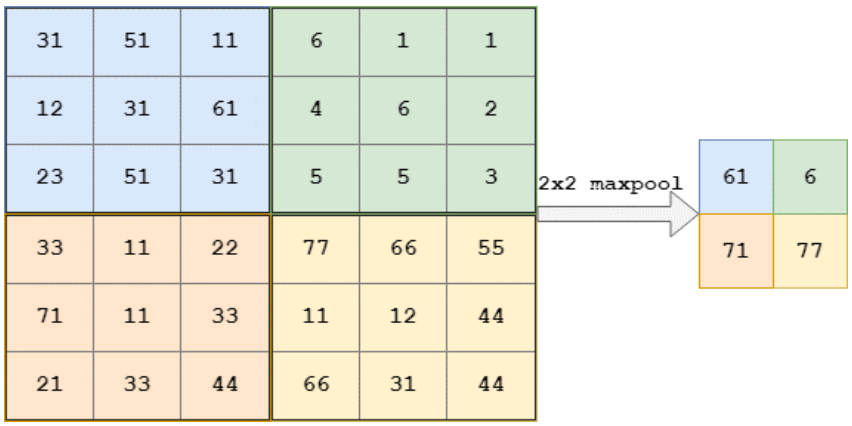
Syviä neuroverkkoja ovat muun muassa konvoluutioneuroverkot ja takaisinkytkennäiset neuroverkot. Usein näissä verkoissa on ennen ulostuloa yksi tiheä kerros, joka muuttaa konvoluutio-osuuden monimutkaisen ulostulon vektoriksi, jossa on yhtä monta alkiota kuin koulutusdatassa luokkia.

Konvoluutioneuroverkot (kuva 4) saavat nimensä niissä olevista konvoluutiokerroksista, joita voi olla yksi tai useampia. Kerrosten välillä on myötäkytkentä syöttökerroksesta kohti ulostulokerrosta, mikä tarkoittaa sitä, että tieto liikkuu ainoastaan ulostulokerrosta kohti.



Kuva 4. Esimerkki konvoluutioneuroverkon rakenteesta [13]

Konvoluutiokerroksessa syötettä suodatetaan vaihtelevan kokoisella suodattimella. Konvoluutiokerroksen jälkeen on usein max-, min-, tai average pooling -kerros, joka käy syötematriisin läpi vaihtelevan kokoisella ikkunalla ja valitsee yhden alkion ulostuloon, maxpool (kuva 5) valitsee ikkunasta suurimman arvoisen alkion, minpool pienimmän ja average laskee kaikkien ikkunoiden alkioiden keskiarvon ulostuloksi.

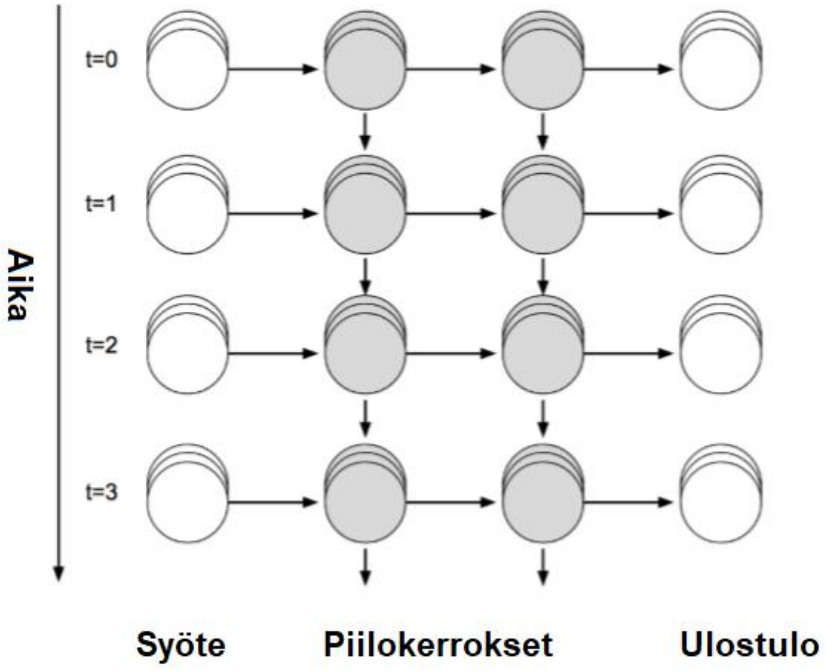


Kuva 5. Esimerkki 2x2 Maxpoolista

Konvoluutio- ja pooling -kerroksia on usein muutamia peräkkäin. Kun syöte liikkuu neuroverkossa eteenpäin, yleensä sen koko X,Y -tasolla pienenee ja Z-tasolla kasvaa. Ennen ulostuloa on usein yksi tiheä kerros softmax-aktivointifunktiolla. Softmax muuttaa jokaisen ulostulon välille $[0,1]$, joka ulostulon todennäköisyyttä.

Takaisinkytkentäiset neuroverkot (eng. Recurrent Neural Network, RNN) erikoistuvat sekvenssien luokitteluun. Niiden pääpiirteenä ovat kerrosten väliset yhteydet, joissa tieto voi liikkua kumpaan suuntaan tahansa, toisin kuin myötäkytketyissä konvoluutioverkoissa. Käytännössä tämä tarkoittaa sitä, verkko säilyttää ajanhetkellä syötetyn syötteen lisäksi myös sitä edeltävien syötteiden sekvenssin. Säilytettyä tietoa ja uutta syötettä voidaan sitten käyttää yhdessä tekemään jokin päätös.

Yleisin takaisinkytkentäisen neuroverkon toteutus on ”pitkä lähimuisti” eli Long Short-Term Memory eli LSTM. Näitä verkkoja käytetään usein puheentunnistustehtävissä, joissa päätöksentekoa tulee viivästyttää, kunnes saatavilla on tarpeeksi tietoa [6]. Seuraavassa kuvassa (kuva 6) on esimerkki LSTM-verkon rakenteesta, jossa syötteellä ajanhetkellä *t* on takaisinkytkentä. Ajanhetkellä $t+1$ on käytössä sekä uusi että edellinen syöte. [13]



Kuva 6. LSTM-verkon rakenne avattuna [13]

1.4 Konenäkö

Konenäössä kuva tai videokehys muutetaan joko päätökseksi tai toisenlaiseksi esitykseksi. Esimerkiksi kuvasta voidaan havaita jotain, jonka perusteella suoritetaan toimenpide, tai havaittua tietoa voidaan esimerkiksi korostaa tai piilottaa. Yleensä konenäköä hyödyntävän järjestelmän kuva- tai videosyötteeseen lisätään metatietoja, esimerkiksi tieto kameran sijainnista kuvattavaa kohdetta kohden. [10]

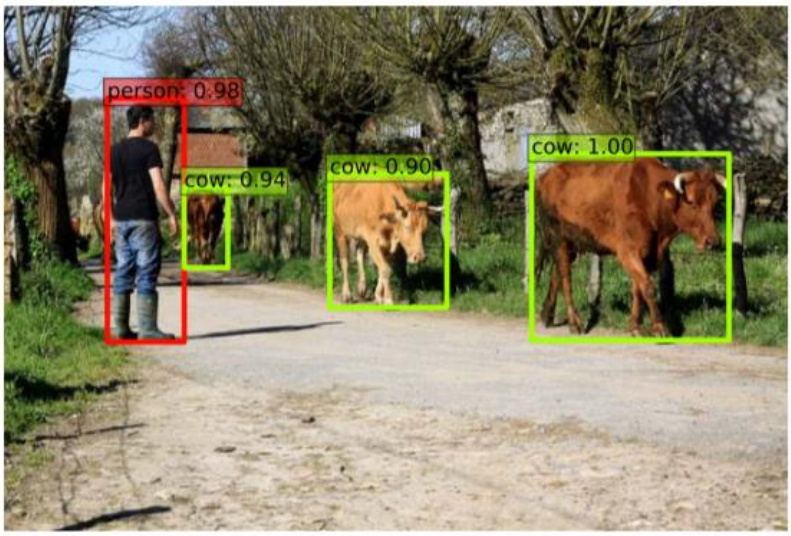
Esiprosessointi on vaihe, jossa kuvalle tai videokehykselle suoritetaan toimenpiteitä, jotka auttavat konenäköjärjestelmän toimintaa. Kuvasta voidaan esimerkiksi tunnistaa ja korostaa reunakohdat (kuva 7). Esiprosessointi ja varsinkin reunojen tunnistus on tärkeä vaihe konenäköjärjestelmissä, sillä ainoastaan reunat sisältävä tai harmaasävyinen kuva on datamäärältään huomattavasti pienempi kuin esiprosessoimaton kuva, joka vaikuttaa sekä konenäköjärjestelmän nopeuteen että sen suorittavan tehtävän luotettavuuteen. [4]

Yleisiä konenäköön liittyviä tehtäviä ovat muun muassa kohteiden tai muotojen tunnistus ja erilaiset kuvien muunnokset, kuten perspektiivin muunnos.

Kuva 7. Reunojen tunnistus

1.5 Objektien tunnistus

Objektien tunnistus on konenäköön ja/tai syväoppimiseen liittyvä ongelma, jonka ratkaisemiseen voidaan käyttää muun muassa syviä neuroverkkoja. Tunnistuksen mahdollistamiseksi vaaditaan koulutusdataa tunnistettavista kohteista kuvissa, joissa tunnistettava kohde on rajattu suorakulmion sisään (kuva 8). Rajauksen lisäksi tarvitaan luokkatieto. Suorakulmion mitat ja objektin luokka muodostavat koulutusdatan, jota kutsutaan myös nimellä *ground truth*. Koulutetulle neuroverkolle syötettyyn kuvaan tai videokehykseen merkitään ulostulossa rajaava nelikulmio ja todennäköisyys sille, että kyseisen nelikulmion sisällä on objekti jostain koulutetusta luokasta, eli varmuus (eng. confidence). [18]

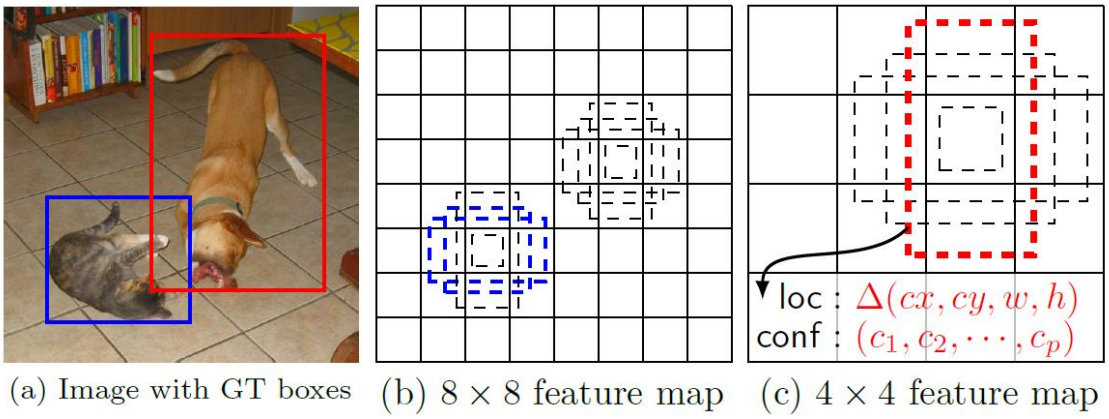


Kuva 8. Kuvasta tunnistettuja kohteita [18]

1.5.1 Kohteentunnistusneuroverkot

SSD eli Single Shot MultiBox Detector on eräs rakenne kohteita tunnistavalle neuroverkolle. Se saa nimensä siitä, että tunnistusta varten syötekuva joudutaan ajamaan vain kerran verkon läpi, mikä tekee siitä erittäin nopean. Nimessä MultiBox viittaa siihen, että koulutusvaiheessa datan jokaiselle luokalle luodaan useita erikokoisia ja muotoisia oletusrajauslaatikoita, jotka oletusrajauslaatikot määrittävät konvoluutiokerrosten syötteiden koot.

Kun verkkoon syötetään syötekuva, konvoluutiokerrosten suodattimet löytävät sille parhaiten sopivimmat oletusrajauslaatikot (kuva 9). Sen jälkeen lasketaan pisteytys, johon kuuluu jokaisen koulutusdatassa olevan luokan varmuus oletusrajauslaatikolle ja sen n:lle siirretylle laatikolle. Ulostulo suodatetaan non-maximum suppression eli NMS-kerroksella, joka pudottaa pois ne rajauslaatikot, joiden varmuus on tietyn rajan alapuolella.

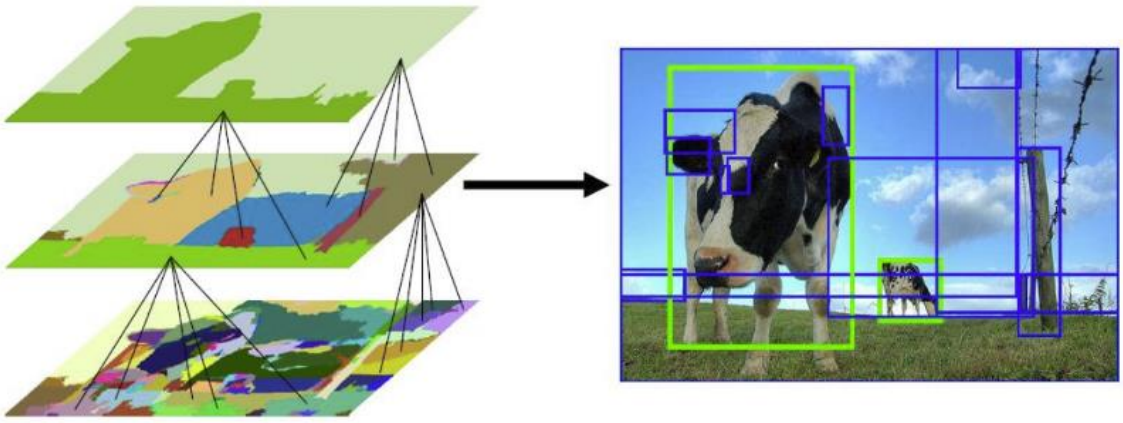


Kuva 9. Single Shot -verkon rajauslaatikot [18]

SSD:n erikokoisten konvoluutiokerrosten ja suodattimien idea perustuu siihen, että eri kohteille sopii paremmin eri kokoiset tai eri orientaatiossa olevat rajauslaatikot. Esimerkiksi ihmiselle sopii paremmin vaakatasossa suurempi nelikulmio, kun taas autolle horisontaalinen nelikulmio. [18]

Region Convolutional Neural Network eli R-CNN on toinen hyvin yleinen kohteiden tunnistukseen käytetty rakenne. Kuten SSD, R-CNN tarvitsee koulutusdataa, johon on merkitty luokkatiedon lisäksi ground truth -rajauslaatikko. Toisin kuin SSD, joka käy syötekuvan läpi vain kerran, R-CNN tekee sen kahdessa vaiheessa.

Ensimmäisessä vaiheessa kuvasta yritetään löytää kiinnostavia alueita käyttämällä jotain region proposal -algoritmia. Region proposal -algoritmit etsivät syötekuvasta alueita, jotka ovat kooltaa, väreiltään ja tekstuureiltaan samankaltaisia. Tuloksena syntyy paljon erikokoisia rajattuja alueita, jotka yhdistetään suuremmiksi alueiksi (kuva 10). Yhdistelyä jatketaan, kunnes jäljellä on noin tuhansissa oleva määrä alueita. Nämä alueet muodostavat ehdotettujen alueiden joukon, joista seuraavassa vaiheessa yritetään etsiä kohteita syötekuvasta.



Kuva 10. R-CNN region proposal -algoritmin ulostulo [16]

Toisesta vaiheesta saadut tunnistustiedot (rajauslaatikot, luokat, varmuudet) voidaan vielä lopuksi suodattaa esimerkiksi NMS:llä tai poistamalla ne rajauslaatikot, joiden IoU toisten rajauslaatikoiden kanssa on liian suuri, toisin sanoen päällekkäisistä rajauslaatikoista vain kaikista varmin jätetään. [16]

1.5.2 Kohteentunnistusmetriikat

Tunnistajien arvioinnissa käytetään useita erilaisia metriikoita. **Tarkkuus** (eng. Precision) kertoo sen, kuinka monta prosenttia löydetyistä kohteista on löydetty oikein:

$$Tarkkuus = \frac{TP}{TP + FP}$$

Herkkyys (eng. Recall) kertoo sen, kuinka monta prosenttia oikeista kohteista on löydetty oikein:

$$Herkkyys = \frac{TP}{TP + FN}$$

Tarkkuus ja herkkyys ovat usein kääntäen verrannollisia. Herkkyyden kasvaessa tunnistaja löytää usein kaikkien oikeiden kohteiden lisäksi myös paljon vääriä kohteita, jolloin TP lähestyy kaikkien kohteiden lukumäärää ja FN lähestyy nollaa, jolloin herkkyyden laskukaava lähestyy maksimiarvoa 1.

Jos herkkyys pienenee, kaikkien löytöjen (TP, FP, FN) lukumäärä vähenee, mutta suurempi osa löydöistä on oikeita (TP), jolloin tarkkuus lähestyy maksimiarvoa 1.

Leikkauksen ja yhdisteen suhteessa (eng. Intersection over Union, IoU) lasketaan tunnistajalla löydetyn rajauslaatikon ja koulutusadatassa olevan rajauslaatikon pinta-alojen leikkauksen ja yhdisteen suhde:

$$IoU = \frac{Rajauslaatikoiden\ leikkaus}{Rajauslaatikoiden\ yhdiste}$$

IoU vaihtelee välillä [0,1] ja se kasvaa, kun tunnistajan löytämän rajauslaatikon koko ja sijainti lähestyvät ground truth -laatikon kokoa ja sijaintia. Kun IoU = 1, tunnistajan löytämä rajauslaatikko on tismalleen sama kuin ground truth -laatikko.

F1-arvo on tarkkuuden ja herkkyyden harmoninen keskiarvo:

$$F1 = \frac{2 * tarkkuus * herkkyys}{(tarkkuus + herkkyys)}$$

Mean Average Precision eli mAP on tunnistusalgoritmin arviointimetriikka, joka vaihtelee välillä [0,1], jossa suurempi on parempi. Sen laskemiseksi täytyy ensin laskea jokaiselle luokalle keskitarkkuus (Average Precision, AP), joiden keskiarvo muodostaa lopullisen mAP-arvon.

Pascal VOC Challenge 2007:n mukaan [5] yhden luokan AP-arvo lasketaan seuraavilla vaiheilla:

1. Tarkkuus ja herkkyys lasketaan erikseen jokaisesta testidatan kuvasta.
2. Lasketut tarkkuudet ja herkkyydet järjestetään järjestetyksi joukoksi niin, että joukon ensimmäisellä alkiolla on suurin tarkkuus ja pienin herkkyys.
3. Herkkyys jaetaan 11:een yhtäsuureen osaan, joista saadaan herkkyysarvojen joukko $R = \{0.0, 0.1, \dots, 1.0\}$
4. AP saadaan laskemalla maksimitarkkuuden keskiarvo herkkyysarvojen joukon R alkioille:

$$AP = \frac{1}{11} \sum_{r \in R} p_{interp}(r)$$

jossa $p_{interp}(r)$ on aikaisemmin lasketuista järjestetyistä tarkkuusarvoista ensimmäinen, jonka vastaava herkkyysarvo on suurempi kuin r.

Kun jokaisen luokan AP on laskettu, mAP saadaan laskemalla niiden keskiarvo. [5]

Tarkkuus, herkkyys, leikkauksen ja yhdisteen suhde ja F1-arvo kuvaavat tunnistajan kykyä löytää kohteet syötteestä. Toinen suorituskykyyn liittyvä piirre on **inferenssinopeus**, joka kertoo sen, kuinka kauan tunnistajalla kesti kohteiden etsimisessä. Inferenssinopeus ilmaistaan kuvien tai videokehysten määrällä sekunnissa eli frames per second tai FPS. [14]

1.6 Perspektiivin muunnos

Perspektiivin muunnos voidaan suorittaa laskemalla kahden kuvan välille homografia. Homografia tarkoittaa suhdetta, jolla kahden eri euklidisessa avaruudessa olevan tasot liittyvät toisiinsa. Toisin sanoen homografialla tarkoitetaan kuvien välistä muunnosta x,y -tasossa, joka voidaan ilmaista matemaattisesti:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

jossa (x,y) ovat kuvan 1 koordinaatteja, (x′,y′) kuvan 2 koordinaatteja ja **H** kuvien välille laskettu homografiamatriisi. Kuvien ei tarvitse olla yhtä suuria, mutta kohdekuvaksi ei kannata valita liian pientä kuvaa, ettei muunnetut pisteet putoa sen ulkopuolelle.

Homografian määritelmän mukaan kahden kuvan $(\boldsymbol{x}, \boldsymbol{y})$ ja $(\boldsymbol{x}, \boldsymbol{y})$**'** väliset pisteet ovat suhteessa toisiinsa kaavalla $(\boldsymbol{x}, \boldsymbol{y})' = \boldsymbol{H} * (\boldsymbol{x}, \boldsymbol{y})$. Toisin sanoen ensimmäisen kuvan pisteen saa muutettua toisen kuvan pisteeksi kertomalla sen x- ja y-koordinaatit homografiamatriisilla ***H***.

Yksinkertaisimmassa suoran lineaarimuunnoksen (eng. Direct Linear Transformation) [8;s.88] tapauksessa homografiamatriisi ***H*** voidaan laskea ratkaisemalla yhtälöryhmä, jossa on kuvien A ja B vastinpisteet. Yksi vastinpistepari P_i muodostaa 2x9 matriisin muotoa:

$$P_i = \begin{bmatrix} -x_A & -y_A & -1 & 0 & 0 & 0 & x_B x_B & y_B x_B & x_B \\ 0 & 0 & 0 & -x_A & -y_A & -1 & x_A y_B & y_A y_B & y_B \end{bmatrix}$$

Muuttujan alaindeksi kertoo, kumman kuvan (A tai B) x- tai y-koordinaatista on kyse.

Homografiamatriisi **H** voidaan ratkaista yhtälöstä $\boldsymbol{Ph} = 0$, jossa $\boldsymbol{h}$ on pystyvektori, joka pitää sisällään tavoitellun matriisin **H** arvot $h_1 \ldots h_9$.

Vastinpistematriisit yhdistetään yhdeksi matriisiksi **Ph**:

$$\boldsymbol{Ph} =$$

$$\begin{bmatrix} -x_{A_1} & -y_{A_1} & -1 & 0 & 0 & 0 & x_{A_1}x_{B_1} & y_{A_1}x_{B_1} & x_{B_1} \\ 0 & 0 & 0 & -x_{A_1} & -y_{A_1} & -1 & x_{A_1}y_{B_1} & y_{A_1}y_{B_1} & y_{B_1} \\ -x_{A_2} & -y_{A_2} & -1 & 0 & 0 & 0 & x_{A_2}x_{B_2} & y_{A_2}x_{B_2} & x_{B_2} \\ 0 & 0 & 0 & -x_{A_2} & -y_{A_2} & -1 & x_{A_2}y_{B_2} & y_{A_2}y_{B_2} & y_{B_2} \\ -x_{A_3} & -y_{A_3} & -1 & 0 & 0 & 0 & x_{A_3}x_{B_3} & y_{A_3}x_{B_3} & x_{B_3} \\ 0 & 0 & 0 & -x_{A_3} & -y_{A_3} & -1 & x_{A_3}y_{B_3} & y_{A_3}y_{B_3} & y_{B_3} \\ -x_{A_4} & -y_{A_4} & -1 & 0 & 0 & 0 & x_{A_4}x_{B_4} & y_{A_4}x_{B_4} & x_{B_4} \\ 0 & 0 & 0 & -x_{A_4} & -y_{A_4} & -1 & x_{A_4}y_{B_4} & y_{A_4}y_{B_4} & y_{B_4} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

Ph on kääntyvä, joten homografiamatriisi **H** voidaan nyt laskea yhtälöllä:

$$\boldsymbol{H} = \boldsymbol{VDV'}$$

jossa **V** sisältää **Ph**:n ominaisvektorit, $\boldsymbol{V'}$ on **V**:n transpoosi ja **D** sisältää ominaisvektoreja vastaavat ominaisarvot. Tuloksena syntyy matriisi, jonka avulla voidaan muuttaa yksi piste kuvasta 1 kuvaan 2, mutta ei toisinpäin. Jos halutaan muuttaa kuvan 2 piste kuvan 1 pisteeksi, pitää sille laskea oma homografiamatriisi. [7]

3. OBJEKTIEN SEURANTA

Objektien tai kohteiden seuranta on konenäköön liittyvä ongelma, joka nousee esille usein valvontaan liittyvissä tapauksissa. Sen tarkoituksena on yhdistää kohteet videokehysten välillä käyttäen hyödyksi jotain kohteen ominaisuutta, esimerkiksi pisteitä tai viivoja. Yleisiä ongelmatilanteita ovat muun muassa kohteen muodon tai valaistuksen muuttuminen tai peittyminen.

Alueisiin perustuvat seuranta-algoritmit (eng. Region-Based tracking) pyrkivät tunnistamaaan rajaavan laatikon, jossa seurattu kohde sijaitsee kehysten välillä. Usein näissä algoritmeissa taustakuvaa käytetään hyväksi seuratun kohteen löytämiseksi. Alueisiin perustuvat seuranta-algoritmit toimivat hyvin tapauksissa, joissa on vähän yksityiskohtia ja huonosti tapauksissa, joissa seurattu kohde peittyy osittain. Joissakin tapauksessa algoritmin tuottama rajaava laatikko voidaan tulkita heikoksi tiedoksi siitä, missä seurattava kohde oikeasti sijaitsee.

Aktiivisiin ääriviivoihin perustuvat seuranta-algoritmit (eng. Contour-Based tracking) erottavat kohteen ääriviivat ja pitävät kirjaa niiden muutoksista kehysten välillä, minkä tuloksena syntyy tarkempaa sijaintietoa kuin alueisiin perustuvissa algoritmeissa. Ne ovat kompleksisuudeltaan yksinkertaisempia kuin alueisiin perustuvat algoritmit ja toimivat myös tilanteissa, joissa kohde peittyy osittain. Sijaintitieto tosin rajoittuu ainoastaan ääriviivoihin, joten kohteen kolmiulotteisen asennon määrittäminen on ongelmallista.

Ominaisuuksiin perustuvat algoritmit (eng. Feature-Based tracking) suorittavat tunnistuksen ja ominaisuuksien, esimerkiksi väritietojen, etsimisen kohteista. Ominaisuudet ryhmitellään ja kohteen uusi sijainti etsitään vertailemalla kuvien ominaisuuksia. Nämä algoritmit jakaantuvat kolmeen alakategoriaan, **paikallisiin**, **globaaleihin** ja **riippuvuusgraafeihin** perustuviin algoritmeihin. **Globaaleissa** algoritmeissa ominaisuuksia toimivat muun muassa keskipisteet, pinta-alat ja värit. Keskipiste voi olla esimerkiksi seuratun kohteen rajauslaatikon keskipiste. **Paikallisissa** algoritmeissa painottuvat viiva- ja käyräsegmentit sekä huiput. **Riippuvusgraafeihin** perustuvissa algoritmeissa käytetään eri ominaisuuksien välisiä etäisyyksiä ja muita geometrisia suhteita. [9]

3.1 Korrelaatiosuodattimet

Signaalinkäsittelyssä suodattimella tarkoitetaan prosessia, joka poistaa tiettyjä taajuuksia tai osia tulosignaalista. Kuvan- ja videonkäsittelyssä suodattimilla voidaan suorittaa muun muassa terävöittämistä ja sumentamista. Suodattimet kuvataan matemaattisesti matriiseina, joita kutsutaan maskeiksi. Esimerkiksi erään sumentamissuodattimen maski voisi olla:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

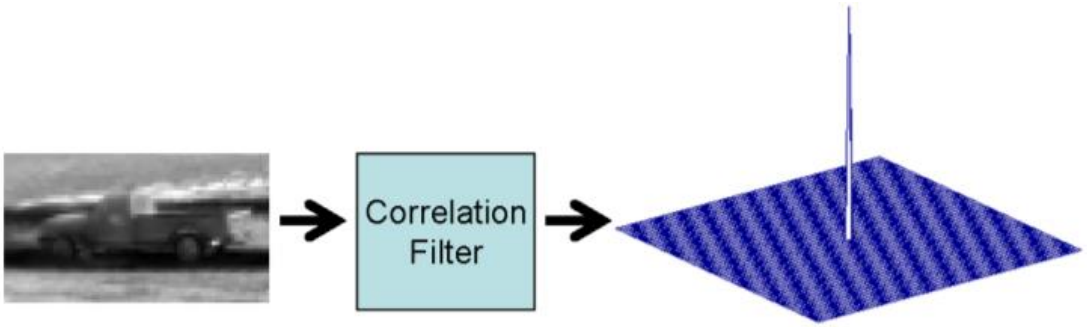
Korrelaatiosuodattimiin perustuvat seuranta-algoritmit muodostavat koulutusvaiheessa suodattimen tai suodattimia, jotka poimivat tietoa seuratusta kohteesta. Kun etsitään kohteen seuraavaa sijaintia, suoritetaan konvoluutio etsintäalueelle ja koulutusvaiheen suodattimille. Tämän konvoluution ulostulo on korrelaatio seuratun kohteen ja etsintäalueen välillä ja suurempi korrelaatio viittaa siihen, että seurattu kohde on todennäköisemmin kyseisellä alueella.

Olkoon $f(x,y)$ tarkasteltava alue syötekuvasta, josta etsitään seurattua kohdetta ja $h(x,y)$ korrelaatiosuodatin (kuva 15, Correlation Filter), joka on koulutettu seurattavalla kohteella. Korrelaatioarvo lasketaan spatiaali- eli x,y -tasossa suorittamalla konvoluutio $f * h$. Olkoon $F(u,w)$ ja $H(u,w)$ kuvan ja suodattimen Fourier-muunnokset. Konvoluutioteorian mukaan:

$$f * h \equiv F(u,w)H(u,w)$$

eli konvoluution Fourier-muunnos voidaan saada myös laskemalla kuvan ja suodattimen Fourier-muunnosten tulo.

Kun seurattu kohde löydetään tarkasteltavalta alueelta, korrelaatioulostuloon syntyy piikki (kuva 11).



Kuva 11. Korrelaatiosuodattimen toiminta [17]

3.1.1 Kernelisöity korrelaatiosuodatin

Kernelisöity kornelaatiosuodatin hyödyntää kernel-menetelmää, jossa syöte muutetaan lineaarikombinaatioiksi käyttämällä jotain kernel-funktiota. Tämä tarkoittaa sitä, että syötteistä tehdään keinotekoisesti moniulotteisempia, mikä saattaa auttaa koulutusvaiheessa parantaen mallin yleistämistä, ilman huomattavia suorituskykyheikennyksiä [3;s.5].

Syötteistä muodostetuista kerneleistä voidaan muodostaa kernelimatriisi *K:*

$$K_{ij} = \kappa(x_i, x_j)$$

jossa κ on kernelfunktio. Yksi esimerkki kernelifunktiosta on d:nnen asteen polynomifunktio, joka määritellään seuraavasti:

$$K(x, y) = (\boldsymbol{x}^T \boldsymbol{y})^d$$

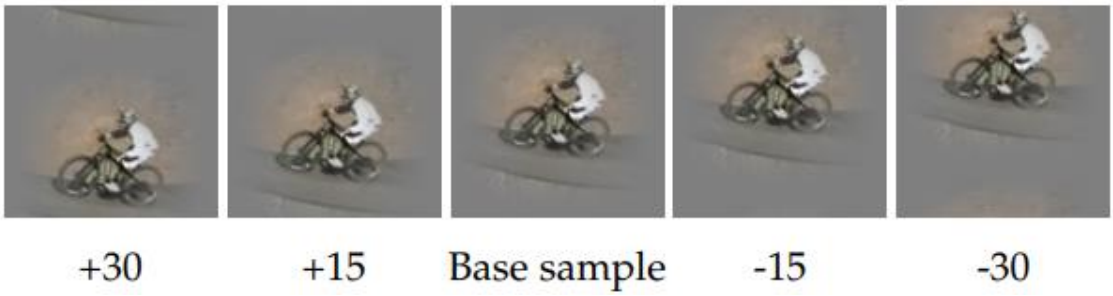
Jossa $\boldsymbol{x}^T$ on syötedatan x-koordinaateista muodostettu pystyvektori ja $\boldsymbol{y}$ on syötedatan y-koordinaatit sisältävä vaakavektori.

Olkoon $\boldsymbol{X}$ n*m matriisi, joka pitää sisällään seuratun kohteen kuvan. Tätä matriisia kutsutaan alkunäyteeksi (base sample). Alkunäytettä käytetään suodattimen koulutukseen siten, että itse alkunäyte on positiivinen näyte ja siitä saadut useat muunnokset eli virtuaaliset näytteet ovat negatiivisia.

Muunnokset voidaan kuvata niin sanotulla syklisellä siirto-operaattorilla, joka on permutaatiomatriisi, eli jokaisella johtavalla rivillä ja sarakkeella on vain yksi 1-arvoinen alkio ja muut nollia:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

Laskemalla matriisituloja $\boldsymbol{P}^u \boldsymbol{X}$ ja muuttamalla *u*:n arvoa saadaan siirtoja, joiden suuruus kasvaa *u*:n kasvaessa. Negatiivinen *u*:n arvo siirtää alkunäytettä vastakkaiseen suuntaan (kuva 12).



Kuva 12. Alkunäyte ja siirrot [8]

Operaattorin syklisyydellä tarkoitetaan sitä, että siirron lopputulokseksi saadaan alkunäyte $\boldsymbol{X}$ n:n siirron välein, joten kaikkien siirtojen joukkoa voidaan merkitä:

$$\{\boldsymbol{P}^u\boldsymbol{X}|u = 0, \dots n-1\}$$

Seuraavaksi tämän joukon kaikki alkiot $\boldsymbol{X}_u$ sijoitetaan matriisiin $\boldsymbol{X}'$, joka muodostaa koulutusdatan. Matriisiin sijoitusta merkitään koulutusdatan funktiona:

$$\boldsymbol{X}' = C(\boldsymbol{X}_u)$$

Merkintä $C(\boldsymbol{X}_u)$ tarkoittaa, että kyseessä on symmetrinen kiertomatriisi, jossa jokainen vaakarivi on käännetty yhden alkion verran oikealle suhteessa edeltävään vaakariviin. Esimerkiksi vaakavektorista $[1 \quad 2 \quad 3]$ muodostettu symmetrinen kiertomatriisi on muotoa:

$$C([1 \quad 2 \quad 3]) = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

Kernelisöity korrelaatiosuodatin ratkaisee koulutusvaiheessa funktion:

$$f(z) = \sum_{i=1}^{n} \alpha_i \kappa(z, x_i)$$

Jossa vektori $\alpha = (K + \lambda I)^{-1} y$ pitää sisällään kertoimet, jotka kuvaavat funktion ratkaisua suljetulla välillä. Koulutusmatriisin $\boldsymbol{X}'$ rivien lukumäärä on sama kuin alkunäytteiden siirtojen lukumäärä eli n-1 ja λ on regularisaatioparametri, jolla pyritään estämään ylisovittaminen. Muuttuja $\boldsymbol{z}$ on alkunäytteen kokoinen matriisi, joka kuvaa tarkasteltavaa aluetta.

Seuraavaksi yritetään löytää seuratun kohteen uusi sijainti. Tarkastellaan useaa eri näytettä $\boldsymbol{Z_i}$ syötekuvasta, sijoitetaan niiden siirtojen $\{\boldsymbol{P}^u\boldsymbol{Z_i}|u = 0, \dots n-1\}$ ja alkunäytteen siirtojen väliset korrelaatiot asymmetriseen kernelimatriisiin $\boldsymbol{K}$, jonka yksi alkio $\boldsymbol{K^z}$ pitää sisällään alkunäytteen siirtojen **X'** korrelaatiot näytteen $\boldsymbol{Z_i}$ siirtojen kanssa:

$$\boldsymbol{K}^z = C(k^{xz})$$

jossa k^{xz} on alkunäytteen ja tarkasteltavan näytteen kernelikorrelaatio, eli matriisi joka kuvaa tarkasteltavan näytteen ja alkunäytteen samankaltaisuutta. Saadaan regressiofunktio, joka diagonalisoituna on muotoa:

$$\hat{f}(z) = \hat{k}^{xz} * \alpha$$

Koska kyseessä on suodatustoimenpide, se suoritetaan taajuustasossa. Yllä olevassa kaavassa hattumerkintä tarkoittaa diskreettiä Fourier-muunnosta. Tuloksena on vektori,

joka pitää sisällään korrelaatioarvot kaikille **yhden** tarkasteltavan alueen siirroille. Kun toimenpide suoritetaan useammalle tarkasteltavalle alueelle, suurin korrelaatioarvo on todennäköisimmin seuratun kohteen uusi sijainti. [8]

3.1.2 Diskriminatiivinen korrelaatiosuodatin

Diskriminatiivisessa korrelaatiosuodattimessa hyödynnetään kanavia eli suodattimia, jotka kuvaavat jotain ominaisuutta, esimerkiksi väri- tai harmaasävytietoa. Kanavien arvot lasketaan sekä koulutuskuvasta että osakuvista, kun yritetään päätellä seuratun kohteen uusi sijainti. Kanavat ovat spatiaalisesti rajoittuneita, eli ne kohdistuvat tiettyyn osaan osakuvasta. Osakuva on kiinnostava alue kuvasta, josta yritetään löytää seuratun kohteen uusi sijainti.

Koska jokainen kanava kuvaa yhtä toisista riippumatonta ominaisuutta, koulutus suoritetaan kanavien kesken riippumattomasti. Koulutuksen tuloksena syntyy spatiaalinen luotettavuuskartta, joka pitää sisällään koulutuskuvan eri osien luotettavuudet kohteen seuraamiseen (kuva 13). Luotettavuuskartan arvot toimivat painokertoimina, kun kanavien korrelaatioarvoja lasketaan. Tämä tarkoittaa sitä, että jos esimerkiksi tietyn värin luotettavuus on hyvä, se vaikuttaa enemmän siihen, mikä sijainti ennustetaan seuratun kohteen uudeksi sijainniksi.

Kuva 13. Koulutuskuva ja spatiaalinen luotettavuuskartta [12]

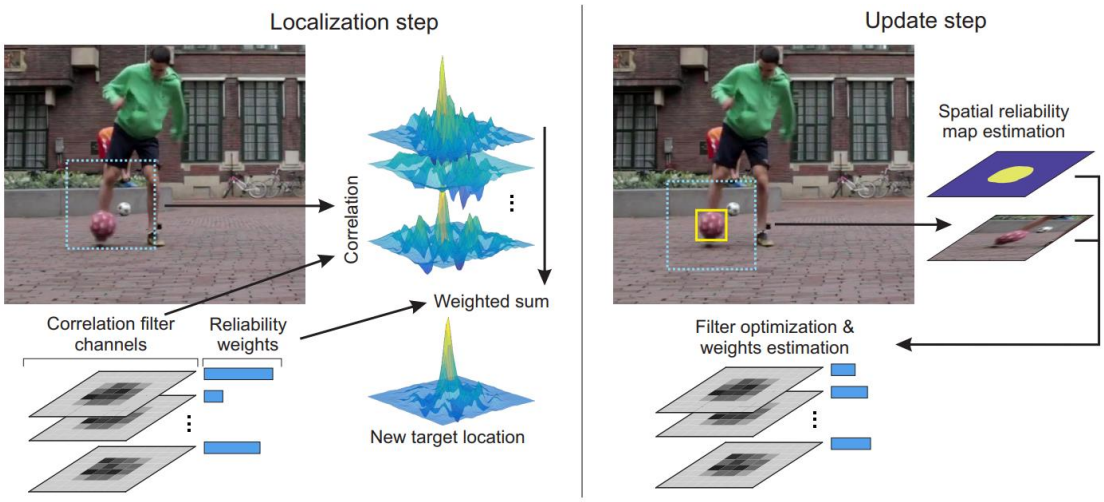
Luotettavuuskartan lisäksi saadaan suodatin ja bittimaski, joka asettaa tiettyjä pikseleitä nolliksi. Jokaiselle kanavalle lasketaan myös ideaalinen korrelaatioarvo, jota käytetään arvioimaan kanavan tuottaman korrelaatioarvon luotettavuutta. Kanavalla on myös toinen luotettavuusarvo, joka lasketaan uutta sijaintia etsiessä ja jonka avulla saatetaan tehdä päätös, että seurattu kohde on kadotettu.

Kun saadaan uusi syötekuva, josta pitää löytää seuratun kohteen uusi sijainti, suoritetaan paikannusvaihe. Olkoon f_d kanavan $d = [1, N_c]$ ominaisuus, joita on N_c kappaletta. Ominaisuudet f_d ovat seuratun kohteen nykyisen sijainnin osakuvan ominaisuuksia ja h_d tarkasteltavan osakuvan ominaisuuksia. Ominaisuudet sisältävät esimerkiksi väri- ja harmaasävytietoja. Seuratun kohteen arvioitu uusi sijainti on siinä kohdassa, missä korrelaatioarvo $g(h)$ on suurin:

$$g(h) = \sum_{d=1}^{N_c} f_d \star h_d$$

Ylläolevassa kaavassa $f_d \star h_d$ tarkoittaa nykyisen sijainnin ja tarkasteltavan alueen osakuvien kanavien välisiä ristikorrelaatiota. Ristikorrelaatio kuvaa kahden signaalin samankaltisuutta, kun toista on siirretty ajanhetken t verran. Sen tarkoituksena on löytää lyhyt signaali pidemmästä signaalista ja tässä tapauksessa tarkasteltavalta alueelta yritetään löytää jokin ominaisuus, joka on opittu koulutuskuvasta eli seurattavasta kohteesta.

Paikannusvaiheen jälkeen koulutusvaihe suoritetaan uudelleen käyttämällä koulutuskuvana keskitettyä uutta sijaintia. Spatiaalinen luotettavuuskartta lasketaan uudelleen ja kanavien maskit estimoidaan käyttäen edellisestä kaavasta saatuja kunkin kanavan korrelaatioarvoja (kuva 14). [12]



Kuva 14. Paikannus- ja päivitysvaihe [12]

3.1.3 MOSSE

Minimum Output Sum of Squared Error eli MOSSE on seuranta-algoritmi, joka perustuu Average of Synthetic Exact Filters (ASEF) -suodattimiin. ASEF-suodatin koostuu koulutuskuvista f_i, niiden tavoitelluista ulostuloista g_i ja tavoiteltuja ulostuloja muodostavista suodattimista h_i. Suodattimet h_i ovat kaksiulotteisia Gaussin suodattimia, jotka keskitetään kiinnostavan kohteen pisteeseen (x_i, y_i) ja joilla on säde σ:

$$h_i(x, y) = e^{\frac{(x-x_i)^2+(y-y_i)^2}{\sigma^2}}$$

Suodatin h_i ratkaistaan kaavasta:

$$h_i(w, v) = \frac{\hat{g}(w, v)}{\hat{f}(w, v)}$$

ASEF-suodatin saadaan summaamalla yhteen suodattimet h_i:

$$H_{ASEF} = \frac{1}{N}\sum_{i=0}^{N} h_i$$

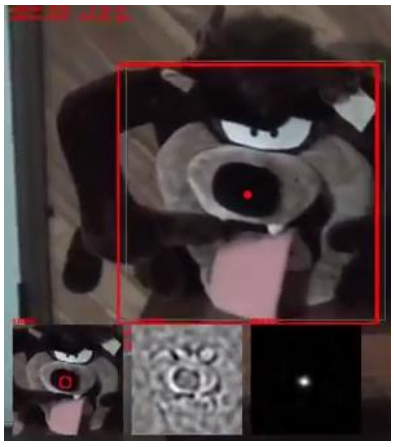
[2]

MOSSE-suodatin lasketaan koulutuskuvista f_i ja ja niiden tavoitelluista suodattimista g_i:

$$H^* = \frac{1}{N}\sum_{i} \frac{\hat{g}_i * \hat{f}_i^*}{\hat{f}_i * \hat{f}_i^*}$$

Kaavassa *-yläindeksi viittaa kompleksikonjugaattiin ja kertomerkki alkioittain tehtävään kertolaskuun. Myös jakolasku suoritetaan alkioittain.

Kuvassa 15 on esimerkki koulutuskuvasta, jossa on merkittynä kiinnostava kohde (vasemmalla), laskettu suodatin H_{ASEF} (keskellä) ja kahden edellisen välinen ulostulo (oikealla).



Kuva 15. MOSSE: Syötekuva, suodatin ja ulostulo [3]

Usein koulutuskuvia on kuitenkin vain yksi, jolloin ASEF- ja MOSSE-suodattimet ovat epäluotettavia. Tämän tilanteen välttämiseksi MOSSE luo keinotekoisesti uusia koulutuskuvia suorittamalla satunnaisia perspektiivin muutoksia syötteenä annettuun koulutuskuvaan.

Seuraamisen aikana kohde saattaa muuttaa muotoaan, valaistustaan tai orientaatiotaan tai se saattaa olla osittain piilossa. MOSSE-algoritmi pyrkii sopeutumaan muutoksiin laskemalla jokaiselle kehykselle uuden suodattimen. Nykyisestä ja kaikista edeltävien kehysten suodattimista lasketaan painotettu keskiarvo, joka painottaa uusimpien kehysten suodattimien vaikutusta.

Seurantavirheiden havaitsemista varten MOSSE laskee Peak to Sidelobe Ratio (PSR) -arvon. Laskemista varten ulostulo g jaetaan kahteen joukkoon, joista ensimmäisessä on sen maksimiarvo (eng. peak) ja toisessa kaikki muut arvot paitsi 11x11 ikkunan arvot maksimiarvon ympäriltä (eng. sidelobe). PSR lasketaan kaavalla:

$$PSR = \frac{g_{max} - \mu_{sl}}{\sigma_{sl}}$$

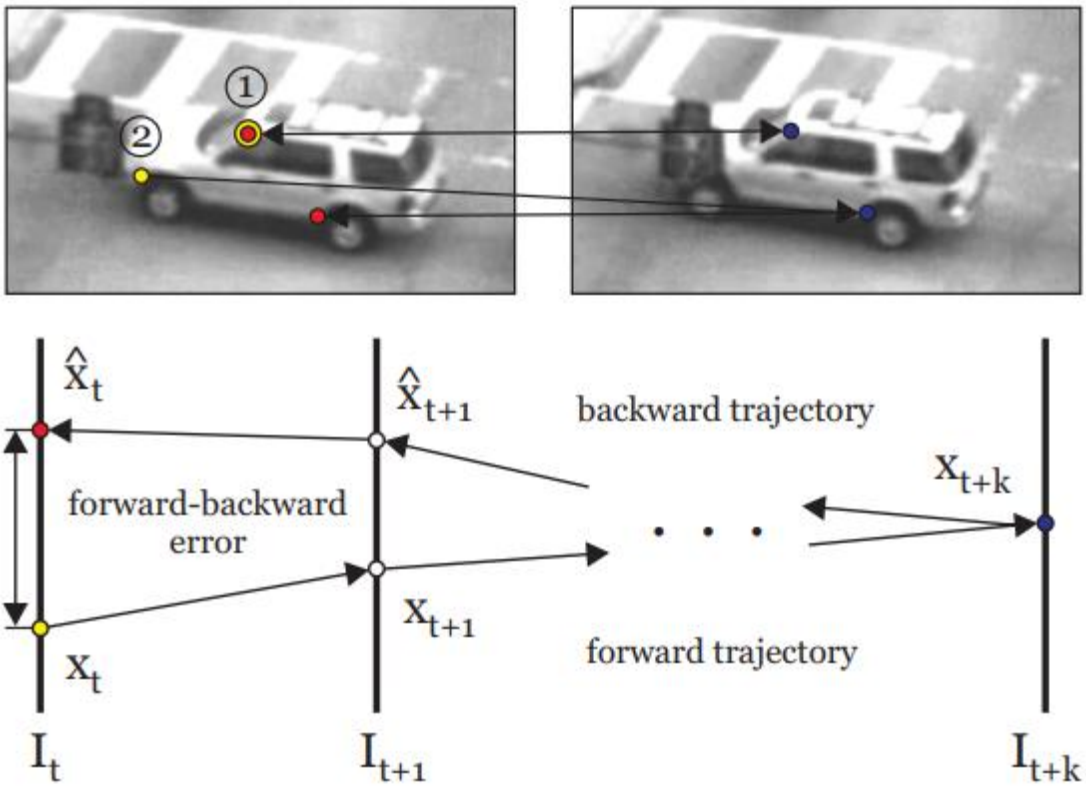
Jossa μ ja σ ovat sidelobe-joukon keskiarvo ja keskihajonta. Onnistuneessa seurannassa PSR:n arvo vaihtelee välillä [20,60]. Jos seurattu kohde peittyy tai seuranta epäonnistuu, $PSR \sim 7.0$. [3]

3.2 MedianFlow-algoritmi

MedianFlow on pisteiden seurantaan perustuva algoritmi. Pisteiden seuranta on yleinen konenäköongelma, jossa estimoimaan pisteen sijaintia ajanhetkellä $t + 1$, kun tiedetään sijainti ajanhetkellä t. Käytännössä on kuitenkin kyse siitä, miten reagoidaan esimerkiksi siihen, että seurattu piste katoaa näkyvistä tai sen muoto muuttuu radikaalisti.

MedianFlow pyrkii havaitsemaan ongelmatilanteet ja arvioimaan omaa luotettavuuttansa niin sanotulla *forward-backward* -menetelmällä, joka perustuu oletukseen, että seuratun kohteen onnistunut seuranta ei ole riippuvainen siitä, mihin suuntaan aika-akselilla liikutaan.

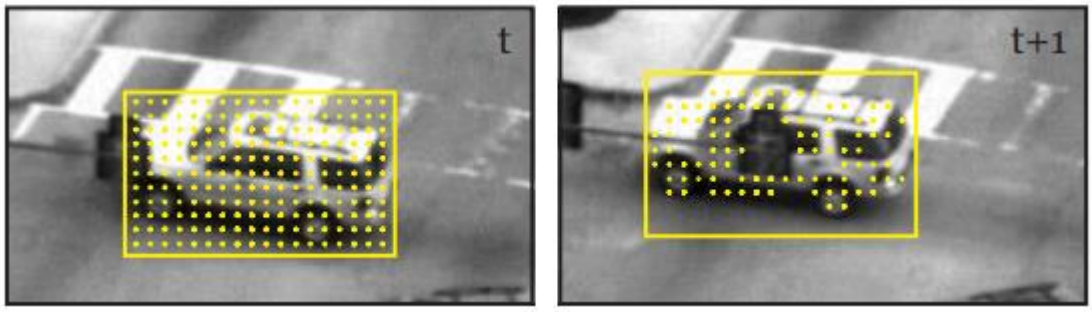
Algoritmi laskee pisteelle liikeradan, jonka suunta on *eteenpäin* aika-akselilla (forward trajectory). Liikeradan päässä olevasta viimeisestä sijainnista lasketaan validointiliikerata, jonka suunta on *taaksepäin* aika-akselilla (backward trajectory). Näitä kahta liikerataa verrataan toisiinsa (forward-backward error, kuva 16) ja jos ero on suuri, niin eteenpäin suuntaava liikerata todetaan virheelliseksi.



Kuva 16. Pisteen liikeratojen estimointi [11]

Koska seurattavat kohteet harvemmin ovat yksittäisiä pisteitä vaan useammista pisteistä koostuvia suurempia kohteita, tulee yhden pisteen seuraamiseen käytettyjä tekniikoita

skaalata ylöspäin, mikä on MedianFlow:n keskeisin idea. Algoritmille syötetään kaksi kuvaa ajanhetkiltä t ja $t + 1$, joista ensimmäinen pitää sisällään seuratun kohteen ja sen rajaavan laatikon B_t. Ulostulona saadaan seuratun kohteen sijainnin rajaava laatikko B_{t+1} ajanhetkellä $t + 1$ (kuva 17).



Kuva 17. Alkukuva ja ennustettu rajauslaatikko [11]

Rajauslaatikkoon B_{t+1} alustetaan joukko pisteitä, joille lasketaan liikeradat käyttämällä Lucas-Kanade -seuranta-algortimia. Liikeratojen laatua mitataan muun muassa aikaisemmin esitetyllä forward-backward -menetelmällä ja puolet huonoimmista liikeradoista suodatetaan. Jäljelle jäävät liikeradat ja niiden pisteet (kuva 19) ajanhetkellä $t + 1$ kuvaavat seuratun kohteen siirtymää. [11]

4. TOTEUTUS

4.1 Testidatan vaatimukset

Sovelluksen testidatana käytettiin lyhyttä videopätkää, jossa on kuvattu ostoskeskuksen avoimella käytävällä käveleviä ihmisiä. Yksinkertaisuuden vuoksi videota on hieman rajattu.

Valvontakameran perspektiivistä kuvattu video on pakollinen. Videon täytyy myös olla vakaa siten, että kuvakulma ei muutu ja ruudunpäivitysnopeus pysyy tasaisena ja realistisena. Koska ohjelman toiminta riippuu videon toistotaajuudesta, tulee sen olla videossa vakaa. Katkokset, hidastumiset ja nopeutumiset videossa voivat aiheuttaa vääristymiä objektien sijaintihistoriassa, mikä taas voi johtaa siihen, että objektille arvattu seuraava sijainti on epärealistisen kaukana tai lähellä.

4.2 Kartoituspisteet

Perspektiivimuunnosta varten kuvalähteeseen ja haluttuun 2-ulotteiseen kohdekuvaan tulee merkitä vastinpisteitä. Seuraavassa kuvassa (kuva 18) esitetään esimerkki vastinpisteistä, jossa kuvalähteenä on käytetty testivideon ensimmäistä kehystä ja kohdekuvana yksinkertaista shakkilautakuviota.

Kuva 18. Vastinpisteiden havainnollistaminen

Vastinpisteet listataan erilliseen tekstitiedostoon, joka ladataan muistiin ohjelman käynnistyessä. Sen jälkeen niiden avulla lasketaan muunnosmatriisi, jonka avulla lähdekuvasta otettu piste voidaan muuttaa kohdekuvan pisteeksi.

4.3 Objektien seuranta

OpenCV on avoimeen lähdekoodiin perustuva konenäköön ja grafiikan käsittelyyn suunniteltu ohjelmointikirjasto. Tässä työssä käytettiin Python-toteutusta version 3.4.2 toteutuksia kappaleessa 3 mainituista seuranta-algoritmeista. Ohjelman suorituksen aikana käytetty algoritmi valitaan ohjelmaan syötettävällä parametrilla. Jokaisen seurattavan kohteen seuraamiseen käytetään ohjelman yhden ajokerran aikana samaa algoritmia.

Algoritmien avuksi kirjoitettiin erillinen luokka seurattavien objektien liikeradan historiaa ja liikeradan ennustusta varten. Kun aloitetaan uuden kohteen seuraaminen, luokasta luodaan uusi instanssi, joka pitää sisällään viittauksen seuranta-algoritmin, eli OpenCV:n toteutuksen instanssista. Tämän viittauksen avulla on helppo päivittää sijaintihistoria, kun kohteelle saadaan uusi sijainti seuraavasta videokehyksestä.

Jokaisella videon kehyksellä seuranta-algoritmi päivitetään. Jos algoritmin päivitys onnistui, saadaan objektin uusi sijainti, joka lisätään seuratun objektin sijaintihistoriaan. Jos päivitys epäonnistuu, objekti poistetaan seurantalistalta.

Yolo on eräs objektien tunnistusta varten rakennettu neuroverkko, josta on saatavilla CPU:ta hyödyntävä toteutus OpenCV-kirjaston lisäosana. Tässä työssä Yoloa hyödynnetään objektien tunnistuksessa, jonka avulla seuranta-algoritmit alustetaan. Ilman objektien tunnistusta työssä käytetyt seuranta-algoritmit vaatisivat käyttäjältä manuaalisen alustustoimenpiteen, jossa seurattavan objektin ympäröivä laatikko valitaan.

Koska Yolo on todettu erittäin hyväksi objektien tunnistuksessa [15], rajataan tunnistuksista pois kaikki muut paitsi kaikista varmimmat löydöt (todennäköisyys >90%). Lisäksi työn rajauksen takia suodatetaan löydöistä pois kaikki muut paitsi ihmiset.

Jokaisella tunnistuskerralla neuroverkko palauttaa myös sellaisia objekteja, joiden seuranta on jo aloitettu. Tällaisia tilanteita varten jokainen neuroverkon löytö tarkistetaan jokaisen jo seurattavana olevan objektin kanssa siten, että lasketaan uuden löydön ja seuratun objektin rajaavien laatikoiden IoU. Jos IoU on suurempi kuin raja-arvo, löytö hylätään.

4.4 Törmäysten ennustus

Ohjelman suoritus voidaan jakaa kahteen osuuteen, datan keräämiseen ja seuraavan sijainnin arvaamiseen kerätyn dataan sovitetun suoran avulla.

4.4.1 Seurantadatan kerääminen

Törmäysten ennustus tehdään samanaikaisesti objektien tunnistuksen kanssa n kertaa sekunnissa, missä n on ohjelmaan syötettävä parametri, esimerkiksi 1.

Ensiksi jokaiselle seuratulle objektille lasketaan liikerata käyttäen pistejoukkoa, joka koostuu seuratun objektin sijaintihistoriassa olevien rajaavien laatikoiden alareunojen keskipisteistä. Tämä päätös liittyy oletukseen, että kameran näkökulmasta ihmisen ”oikea” sijainti on aina lähimpänä rajaavan laatikon alareunan keskiosaa. Seuraavassa kuvassa on esimerkki videokuvan osasta, jossa ihmisen rajaavan laatikon alareunan keskelle on merkitty punaisella se piste, jota tullaan käyttämään liikeradan laskemiseen (kuva 19).

Kuva 19. Ihmisen rajaava laatikko ja sijaintipiste

4.4.2 Seuraavan sijainnin ennustaminen

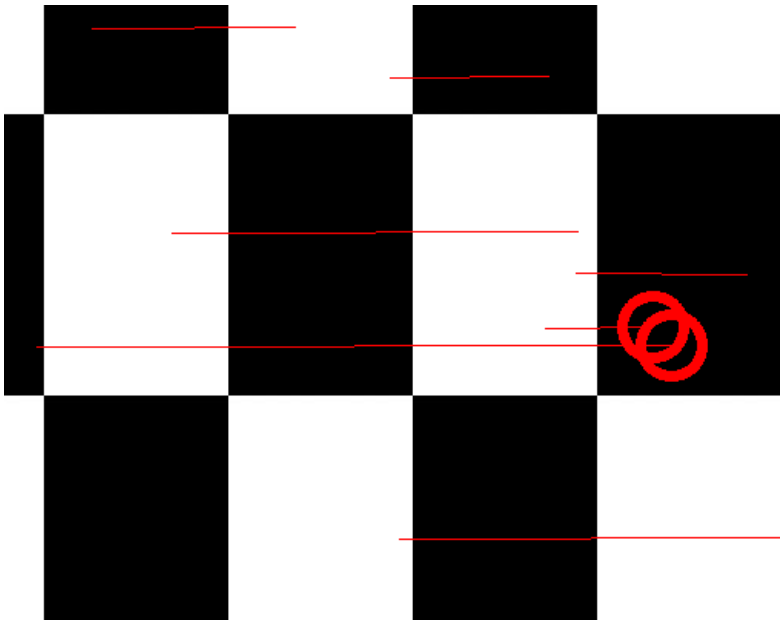
Kun pistejoukko on selvillä, sovitetaan siihen suora. Ratkaistaan siis yhtälöstä $ax + b$ kerroin a ja vakio b. Sovitettua suoraa jatketaan lisäämällä tai vähentämällä viimeisestä x-koordinaatista seuratun objektin laskettu nopeus riippuen siitä, mihin suuntaan seurattu objekti on liikkumassa. Liikesuuntaa voidaan arvioida tarkastelemalla edellisten pisteiden y-koordinaatteja ja laskettua kerrointa a, joka on kulmakerroin.

Nopeutta arvioidaan tutkimalla etäisyyttä, jonka seurattu objekti on liikkunut viimeisen sekunnin aikana koordinaatistossa.

Kun suora on sovitettu ja jatkettu yhdellä pisteellä, piirretään siihen ympyrä, jonka säde x = ”henkilökohtainen etäisyys” on ohjelmaan syötettävä parametri. Tämä ympyrä on seuratun objektin arvioitu sijainti sekunnin kuluttua.

Seuraavaksi jokaisen seuratun objektin arvioitu sijaintiympyrä tarkistetaan toisten objektien kanssa siten, että jos ympyrät leikkaavat, on kyseessä mahdollinen törmäys.

Kuvassa 20 on esimerkki kohdekuvasta, johon on piirretty kaikkien viivoja, jotka kuvaavat seurattujen kohteiden liikkeitä nykyisestä sijainnista arvattuun sijaintiin. Mahdollisessa törmäystapahtumassa kuvaan piirretään myös ympyrät, jotka kuvaavat törmänneiden kohteiden henkilökohtaisia alueita.



Kuva 20. Kohdekuvaan piirretty törmäystapahtuma

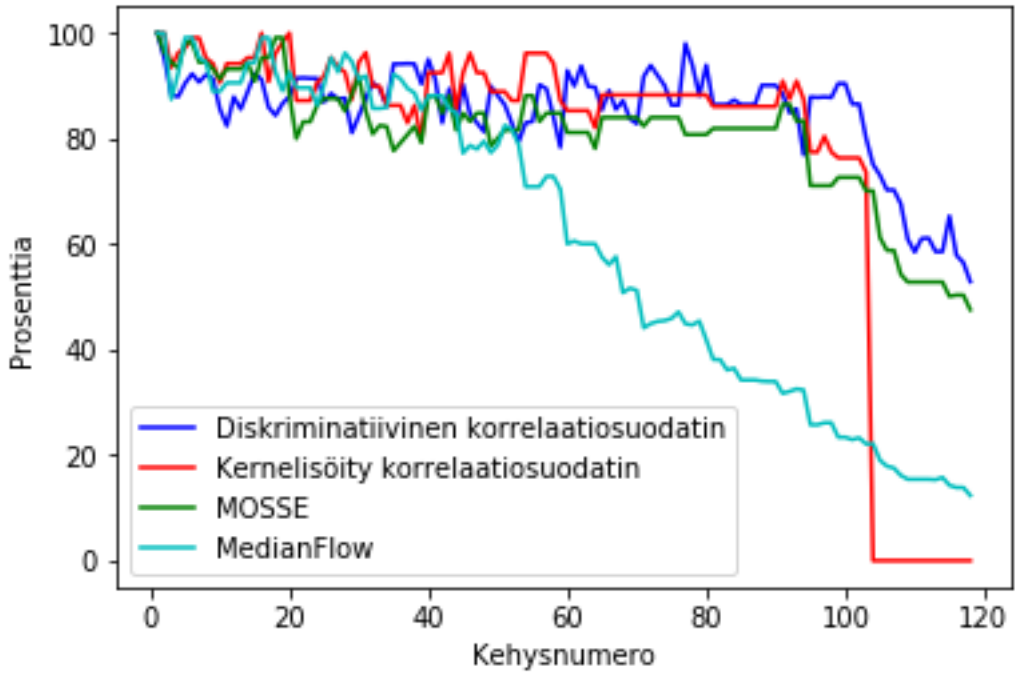
5. TULOKSET

5.1 Seuranta-algoritmien vertailu

Seuraavaksi esitellään seuranta-algoritmien testituloksia. Jokaiselle seuranta-algoritmille näytetään sama videosekvenssi (kuva 21), josta seurataan yhtä kohdetta. Video on kuvattu valvontakameranäkökulmasta ja seurattu kohde liikkuu kävelyvauhtia poispäin kameraan nähden. Sekvenssin lopussa seurattu kohde poistuu osittain kuvattavalta alueelta.

Kuva 21. Testivideosekvenssin 1., keskimmäinen ja viimeinen kehys

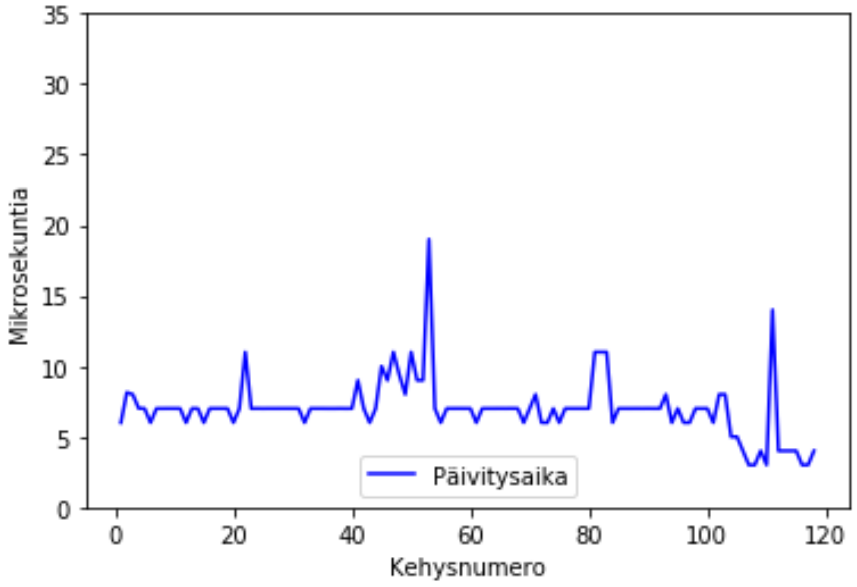
Kuvassa 22 on kuvattuna kaikkien seuranta-algoritmien IoU-arvot videosekvenssin kehysnumeron funktiona.



Kuva 22. Seuranta-algoritmien IoU testisekvenssissä

5.1.1 Kernelisöity korrelaatiosuodatin

Kernelisöity korrelaatiosuodatin on erittäin nopea (kuva 23), mutta tietyissä tilanteissa epäluotettava algoritmi. Esimerkiksi kuvasta poistuvat kohteet eivät aina aiheuta virhetilannetta, joka johtaa siihen, että algoritmi jatkaa kohteen viimeisimmän sijainnin seuraamista paikallaan.



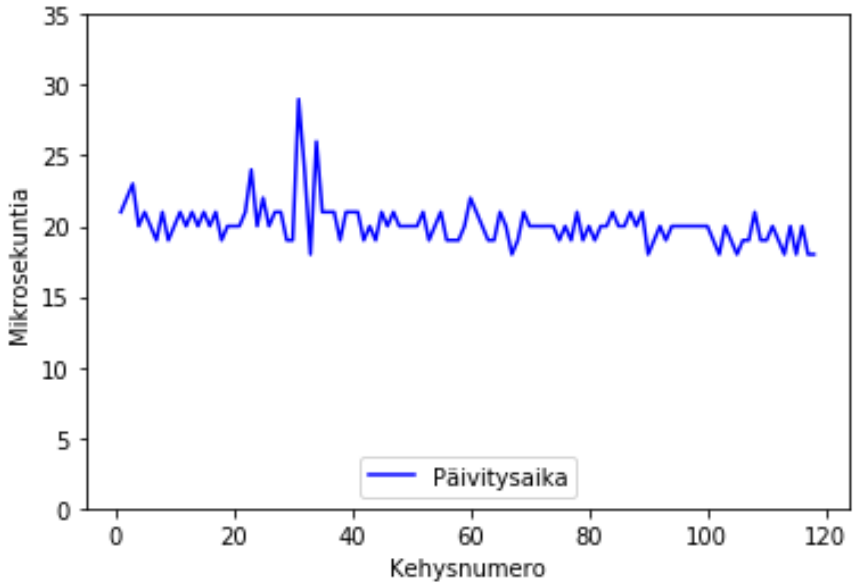
Kuva 23. Kernelisöity korrelaatiosuodatin, päivitysnopeus

Vielä pahemmaksi tilanteen tekee se, että toinen seurattava kohde liikkuu jumittuneen vierestä, jolloin jumittunut seurantainstanssi alkaa seurata sitä. Tämä on erittäin huono tilanne törmäysten ennustuksen kannalta, sillä nyt yhtä kohdetta seuraa kaksi seurantainstanssia. Kun seuratun objektin seuraavaa sijaintia arvataan parin sekunnin liikkumisen jälkeen, tulee väistämättä sama sijainti kahdelta seurantainstanssilta, joka virheellisesti aiheuttaa törmäyshälytyksen.

Videosekvenssin lopussa seuratun kohteen poistuessa kuvasta, kernelisöity korrelaatiosuodatin kadottaa kohteen, jolloin IoU putoaa nollaan.

5.1.2 Diskriminatiivinen korrelaatiosuodatin

Diskriminatiivinen korrelaatiosuodatin on luotettava ja tarkka, mutta sen päivittämiseen kuluu lähes kolminkertainen aika muihin algoritmeihin verrattuna (kuva 24). Seurannan aikana ennustetun rajauslaatikon koko pysyy lähes muuttumattomanä.

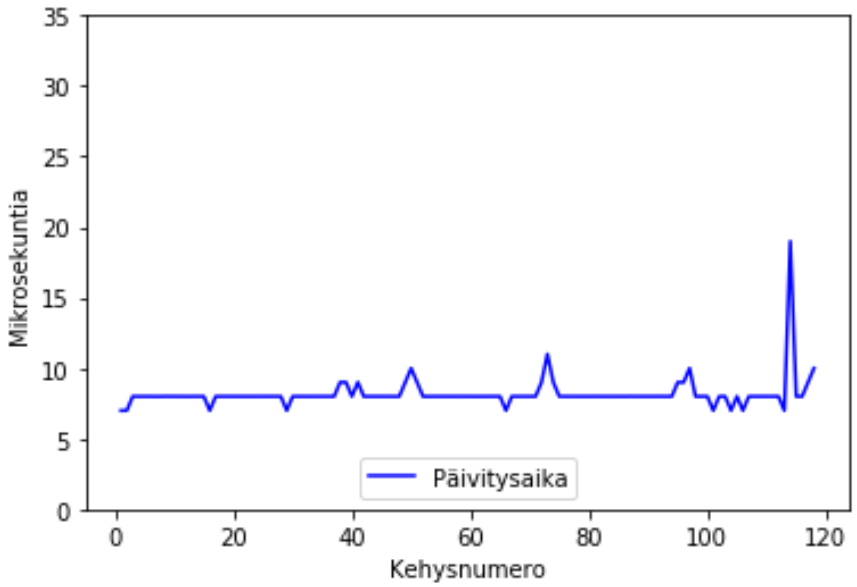


Kuva 24. Diskriminatiivinen korrelaatiosuodatin: päivitysnopeus

Seuratun kohteen liikesuunta tai nopeus eivät vaikuta tarkkuuteen. Myöskään kohteen hetkellinen osittainen peittyminen ei aiheuta ongelmia algoritmille. Kun kohde poistuu kuvasta, algoritmi tunnistaa sen virhetilanteeksi, jolloin seuraaminen voidaan lopettaa.

5.1.3 MedianFlow-algoritmi

Tämä algoritmi on nopeudessa (kuva 25) ja tarkkuudessa keskitasoa. Luotettavuus on kyseenalaista tapauksissa, jossa seurattavan kohteen koko muuttuu. Koon muutosta tapahtuu silloin, kun seurattu kohde liikkuu kohti kameraa tai poispäin kamerasta. Näissä tapauksissa tämä seuranta-algoritmi saattaa jäädä jumiin lähelle kohtaa, jossa seuranta aloitettiin.

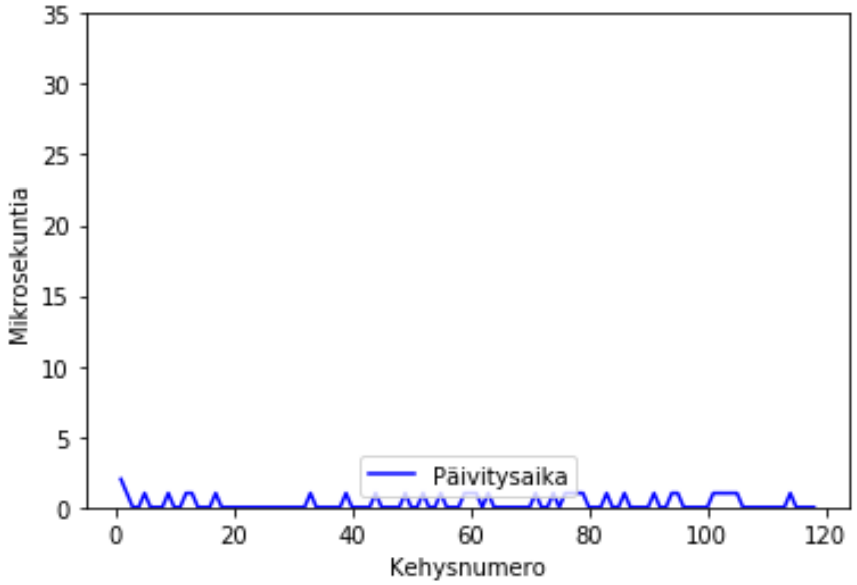


Kuva 25. MedianFlow: päivitysnopeus

Muuttuvan kokoisen kohteen tapauksessa ennustettu rajauslaatikko kasvaa samalla, kun kohteen koko muuttuu. Tämä johtaa siihen, että mitattu IoU-arvo lähestyy nollaa. Koska ennustettu laatikko on suuri, seuratun kohteen oikeasta sijainnista ei ole luotettavaa tietoa, mikä tekee törmäysten ennustamisesta mahdotonta.

5.1.4 MOSSE

Ylivoimaisesti nopein (kuva 26) ja kaiken lisäksi erittäin luotettava. Seuratun kohteen rajauslaatikko säilyttää kokonsa ja pysyy tiukasti kohteessa. Ground truth -rajauslaatikko ja ennustettu rajauslaatikko pysyvät lähes samankokoisina, mutta loppua kohden seuratun henkilön poistuessa osittain kuvasta ennustettu rajauslaatikko kasvaa hieman.



Kuva 26. MOSSE: päivitysnopeus

6. YHTEENVETO

Tässä diplomityössä kehitettiin yksinkertainen Python-sovellus, jonka tehtävänä on tunnistaa syötevideosta ihmiset, seurata heitä ja tietyn väliajoin ennustaa heidän seuraavat sijaintinsa. Ennustettujen sijaintitietojen mukaan yritetään päätellä, onko joku seuratuista ihmisistä mahdollisesti törmäämässä toiseen seurattuun ihmiseen. Sovellusta testattiin videolla, jossa on kuvattu ylhäältäpäin ostoskeskuksen avoimella alueella käveleviä ihmisiä.

Ihmisten tunnistus suoritettiin käyttämällä Yolo-neuroverkon OpenCV:n tarjoamaa toteutusta. Valinta perusteltiin sillä, että Yolon suorituskyky on todistetusti kohteita tunnistavien neuroverkkojen parhaimmistoa sekä tarkkuudessa että nopeudessa [15]. OpenCV:n tarjoama toteutus mahdollisti helpon käyttöönoton ja kohtalaisen suorituskyvyn jopa CPU:lla.

OpenCV tarjoaa useita toteutuksia kohteiden seuraamiselle, joista tässä työssä tutkittiin diskriminatiivista korrelaatiosuodatinta, kernelisöityä korrelaatiosuodatinta, MedianFlow:ta sekä MOSSE:a.

Diskriminatiivinen korrelaatiosuodatin oli testeissä tarkin, mutta hitain seuranta-algoritmi. Sen tarkkuuteen ei vaikuttanut seuratun kohteen liikkumisnopeus, kulkusuunta kameraan nähden tai osittain peittyminen. Päivitykseen eli kohteen löytämiseen uudesta kuvasta kului kolminkertainen aika muihin algoritmeihin verrattuna.

Kernelisöity korrelaatiosuodatin on nopea mutta virheherkkä. Sille tuottaa ongelmia osittain peittyvät kohteet, esimerkiksi ihmiset, jotka kulkevat toistensa ohi. Tälläisessä tapauksessa algoritmi saattaa aloittaa virheellisesti seuraamaan ohi kulkenutta ihmistä tai jäädä jumiin siihen kohtaan, jossa seurattu kohde peittyi.

MedianFlow on yhtä nopea kuin kernelisöity korrelaatiosuodatin, mutta paljon epäluotettavampi. Sille tuottaa ongelmia kohteet, joiden valaistus tai muoto muuttuu. Esimerkiksi tilanteessa, jossa seurattu kohde kulkee poispäin kameraa kohden, MedianFlow:n ennustama kohteen rajauslaatikko kasvaa kohteen kulkusuuntaan kohti, minkä takia kohteen sijainnista ei ole luotettavaa tietoa.

MOSSE oli testeissä ylivoimaisesti nopein ja tarkkuudessa vain hivenen huonompi kuin tarkin algoritmi, diskriminatiivinen korrelaatiosuodatin. Kohteen liikesuunta, nopeus tai peittyminen eivät vaikuttaneet tarkkuuteen, mutta peittyessä ennustettu rajauslaatikko saattoi kasvaa hieman, mikä heikentää kohteen sijaintitiedon luotettavuutta.

Algoritmien testituloksista voidaan päätellä, että MOSSE-seuranta-algoritmin nopeus ja tarkkuus ovat huippuluokkaa, ainakin testisekvenssissä. Diskriminatiivinen korrelaatiosuodatin on hieman luotettavampi tarkkuudeltaan, mutta huomattavasti hitaampi. Voidaan päätellä, että kohteiden seuraamiseen paras vaihtoehto on diskriminatiivinen korrelaatiosuodatin tapauksessa, jossa seurattavia kohteita on suhteellisen vähän (<10) ja jossa tarkkuus on tärkeämpää kuin nopeus. Jos kohteita on enemmän, suodattimen päivitysnopeuden hidastuminen alkaa vaikuttaa niin paljon, että jopa tunnistusten tekeminen neuroverkolla CPU:lla on nopeampaa. Tällöin MOSSE on ehdottomasti parempi vaihtoehto. Koska diplomityön alkuperäinen idea tuli työturvallisuudesta ja valvontakamerakuvan tutkimisesta, tarkkuus ja varmuus ovat korkeimmat prioriteetit.

Sovellukseen kehitettiin yksinkertainen törmäysten ennustusalgoritmi, joka etsii seurattujen kohteiden ennustetuista sijainneista päällekkäisyyksiä. Päällekkäisyys määriteltiin henkilökohtaisella tilalla, joka on ympyrä. Kohteen ennustettua sijaintia käytettiin ympyrän keskipisteenä ja säde saatiin ohjelman käynnistysparametrina. Jos ympyrät leikkasivat, tapahtuma tulkittiin törmäykseksi. Testivideossa pystyttiin testaamaan algoritmin toimivuutta, mutta sen luotettavuutta on mahdotonta arvioida yhden videon perusteella. Koska käyttökelpoista valvontakamerakuvaa on lähes mahdotonta löytää, algoritmin luotettavuus on kyseenalainen.

Sovelluksen jatkokehityksessä ensimmäisen prioriteetin tulisi olla rinnakkaistaminen. Tällä hetkellä koko sovellus ajetaan yhdessä säikeessä, joka on suuri pullonkaula varsinkin seuraavien sijaintien ja törmäysten ennustamisessa. Yolo-neuroverkkoa ajettiin CPU:lla, mikä aiheutti videokuvan käsittelyn pysähdyksen noin sekunnin ajaksi. OpenCV:n tarjoama toteutus on helppo ottaa käyttöön, mutta se ei tue GPU:ta, joten olisi erittäin hyödyllistä ladata Yolo ja kääntää se GPU:lla suoritettavaan muotoon.

Alkuperäinen idea oli tutkia valvontakameran kuvaa, mutta työ rajattiin videoihin. Toteutusta pitäisi siis muuttaa hieman niin, että videopolun sijaan sovellus saisi käynnistysparametrina esimerkiksi verkkoon kytketyn valvonta- tai webkameran IP-osoitteen. Kameroista saatava videokuva ei kuitenkaan ole mp4-muodossa, joten videokuvan vastaanottamiseksi pitäisi lisätä jokin videovirran vastaanottamiseen ja puskurointiin kykenevä toteutus.

Seuratun kohteen seuraava sijainti ennustettiin käyttämällä edellisiä sijainteja sekunnin ajalta ja sovittamalla niihin suora. Suoraa jatkettiin x-akselilla joko positiiviseen tai negatiiviseen suuntaan riippuen sovitetun suoran kulmakertoimesta ja edellisten pisteiden y-koordinaattien suunnasta (kasvava, vähenevä). Ennustukset olivat yleisesti realistisia, mutta algoritmi on erittäin herkkä videon nopeuden vaihteluille ja ihmisten äkkisille suunnanvaihdoille. Ennustusalgoritmia olisi siis hyvä kehittää sietämään paremmin tälläisiä tilanteita, esimerkiksi keskiarvottamalla edellisiä sijainteja tai sovittamalla suoran sijaan toisen asteen yhtälön.

LÄHTEET

[1] E. Barrett – In China, Facial Recognition Technology is Watching You http://fortune.com/2018/10/28/in-china-facial-recognition-tech-is-watching-you/

[2] D. S. Bolme, B. A. Draper and J. R. Beveridge, "Average of Synthetic Exact Filters," *2009 IEEE Conference on Computer Vision and Pattern Recognition,* Miami, FL, 2009, pp. 2105-2112.

[3] Bolme, David S., J. Ross Beveridge, Bruce A. Draper and Yui Man Lui. “Visual object tracking using adaptive correlation filters.” *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2010)*: 2544-2550.

[4] J. Canny, "A Computational Approach to Edge Detection," *in IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[5] Everingham, M., Van Gool, L., Williams, C.K.I. et al. ” The Pascal Visual Object Classes (VOC) Challenge” in *Int J Comput Vis* (2010) 88: 303. https://doi.org/10.1007/s11263-009-0275-4

[6] Fernández S., Graves A., Schmidhuber J. (2007) “An Application of Recurrent Neural Networks to Discriminative Keyword Spotting.” In: *de Sá J.M., Alexandre L.A., Duch W., Mandic D. (eds) Artificial Neural Networks – ICANN 2007. ICANN 2007. Lecture Notes in Computer Science,* vol 4669. Springer, Berlin, Heidelberg

[7] R. Hartley – Multiple View Geometry in computer vision (Cambridge University Press, 2014)

[8] J. F. Henriques, R. Caseiro, P. Martins and J. Batista, "High-Speed Tracking with Kernelized Correlation Filters," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583-596, 1 March 2015.

[9] Weiming Hu, Tieniu Tan, Liang Wang and S. Maybank, "A survey on visual surveillance of object motion and behaviors," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 34, no. 3, pp. 334-352, Aug. 2004

[10] A. Kaehler, G. Bradski – Learning OpenCV 3 (O’Reilly, 2017)

[11] Z. Kalal, K. Mikolajczyk and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," *2010 20th International Conference on Pattern Recognition*, Istanbul, 2010, pp. 2756-2759.

[12] A. Lukežič, T. Vojíř, L. Čehovin, J. Matas, M. Kristan – Discriminative Correlation Filter Tracker with Channel and Spatial Reliability

[13] J. Patterson, A. Gibson – Deep Learning: A Practitioner’s Approach (O’Reilly, 2017)

[14] Powers, David M. W.. “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation.” (2008).

[15] J. Redmon, A. Farhadi – YOLO9000: Better, Faster, Stronger

[16] S. Ren, K. He Ross, G. Jian Sun – Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

[17] Andres Rodriguez, Vishnu Boddeti, BVK Vijaya Kumar, Abhijit Mahalanobis, “Maximum Margin Correlation Filter: A new approach for localization and classification” in *IEEE Transactions on Image Processing* (TIP) 2013.

[18] Liu W. et al. (2016) “SSD: Single Shot MultiBox Detector.” In: *Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science*, vol 9905. Springer, Cham